The Dissertation Committee for Timothy Matthew Josserand Certifies that this is the approved version of the following dissertation:

Optimally-Robust Nonlinear Control of a Class of Robotic Underwater Vehicles

Committee:

Benito Fernandez, Supervisor

Joseph Beaman

Terry Henderson

Keith Lent

Raul Longoria

Optimally-Robust Nonlinear Control of a Class of Robotic Underwater Vehicles

by

Timothy Matthew Josserand, B.S.; M.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

December 2006

Dedication

I dedicate this dissertation to Theresa for helping make this achievement possible.

Acknowledgements

I would like to acknowledge Dr. Benito Fernandez for the many years of excellent guidance and instruction in the field of nonlinear controls and intelligent engineering methods. I also thank the members of my committee for their time and invaluable feedback. Finally, I would like to thank my colleagues in the Applied Research Labs at the University of Texas at Austin for their support and encouragement.

Optimally-Robust Nonlinear Control of a Class of Robotic Underwater Vehicles

Publication No._____

Timothy Matthew Josserand, Ph.D.

The University of Texas at Austin, 2006

Supervisor: Benito Fernandez

The subject of this dissertation is the optimally-robust nonlinear control of a class of robotic underwater vehicles (RUVs). The RUV class is characterized by high fineness ratios (length-to-diameter), axial symmetry, and passive roll stability. These vehicles are optimized for robotic applications needing power efficiency for long-range autonomous operations and motion stability for sensor performance improvement. A familiar example is the REMUS vehicle.

The particular robot class is further identified by an inconsistent actuator arrangement where the number of inputs is fewer than the number of degrees of freedom, by the loss of controllability at low surge speeds due to the use of fin-based control actuation, and by an inherent heading instability. Therefore, this important type of RUV comprises an interesting and challenging class of systems to study from a control theoretic perspective.

The optimally-robust nonlinear control method combines sliding mode control with stochastic state and model uncertainty estimation. First a regular form sliding mode control law is developed for the heading and depth control of the RUV class. The Particle Filter algorithm is then modified and applied to the particular case of estimating not only the RUV state for control feedback but also the functional uncertainty associated with partially modeled shallow water wave disturbances. The functional uncertainty estimate is used to dynamically adjust the sliding mode controller performance term gain according to the estimate of the wave phase and the RUV's orientation with respect to the predominate wave direction. As a result, the RUV experiences increased performance over constant gain and Kalman Filter methods in terms of heading stability which increases effectiveness and decreased actuator power consumption which increases the RUV mission time. The proposed technique is general enough to be applied to other systems.

An experimental RUV was designed and constructed to compare the performance of the regular form sliding mode controller with the conventional PID-type controller. It is demonstrated that the more complicated formulas of the regular form sliding mode controller can still be implemented real-time in an embedded system and that the controller's performance with regard to modeling uncertainty justifies the added complexity.

Table of Contents

List of Tables ..... **x**

List of Figures ..... **xi**

Chapter 1: Introduction **1**

Dissertation Contribution 5

Dissertation Organization 7

CHAPTER 2: LITERATURE REVIEW **9**

Introduction 9

RUV Model 9

Nonlinear Sliding Mode Control 10

Optimal Gain Selection and Nonlinear State Estimation 11

Chapter 3: Mathematical Model of the RUV Class **13**

Vehicle Description 13

Coordinate System and State variables 13

Coordinate Transformation Matrix and Vehicle Kinematics 15

Derivation of the Complete RUV Equations of Motion 17

External Forces and Moments 21

Model Testing 46

Chapter 4: RUV Sliding Mode Control **73**

Introduction 73

Stability Considerations for RUVs with Inconsistent Actuator Configurations 73

Mathematical Preliminaries 77

Regular Form Sliding Mode Control 83

Application to the RUV model 87

Regular Form Sliding Mode Control Results 98

Chapter 5: Experimental RUV **104**

Introduction 104

Vehicle Description 104
Comparison: PID vs. Sliding Mode Embedded Control....113

Chapter 6: Optimal Gain Sliding Mode Control 123
Optimal Sliding Mode Control Concept....123
Obtaining the Optimal Uncertainty Estimate....125
Optimally-Robust Control of Robotic Underwater Vehicle 146
Results....154
discussion....172

Chapter 7. Conclusions and Recommendations....175
Summary....175
Dissertation Contribution....176
Recommendations....177

Appendix....180
Experimental RUV Schematics 180
Experimental RUV Code Listing....182

References....216

Vita 222

List of Tables

Table 3.1. REMUS RUV Simulation Parameters 48
Table 3.2. REMUS Moment Coefficients 49
Table 3.3. REMUS Force Coefficients 50
Table 5.1 RUV Primary Components Parts Listing 106
Table 5.2. Experimental RUV Parameters 116
Table 5.3. Experimental RUV Moment Coefficients 117
Table 5.4. Experimental RUV Force Coefficients 118
Table 6.1. The Extended Kalman State and Gain Observer 131
Table 6.2. The Particle Filter State and Gain Observer 134
Table 6.3. The Particle Filter State and Gain Observer with Sampled Process Dynamics 143
Table 6.4. Results Comparison Between the Constant Gain (CG), the Extended Kalman Filter (EKF) and Particle Filter (PF) Optimally-Robust Controllers 172
Table 6.5. Particle Number Results Comparison 173

List of Figures

Figure 1.1. Examples of the RUV class described in the text. Top: The University of Southampton's Autosub. Bottom: Woods Hole Oceanographic Institute's modular RUV, the REMUS 100. The illustration demonstrates the REMUS's configuration modularity. 2
Figure 3.1. Earth-fixed and body-fixed coordinate systems. The RUV center of gravity is shown as a red dot. The spinning arrows indicate sense of rotation of the Euler angles (φ, θ, ψ) and the body-fixed angular momentum vector $\Omega = (p, q, r)$. The linear velocity vector of the RUV is (u, v, w). The location of the RUV reference frame origin with respect to the Earth-fixed reference frame origin is r_o. The location of the RUV center of gravity with respect to the Earth-fixed reference frame origin is r_e. The RUV center of gravity is fixed at r_g with respect to the RUV frame. Finally, the components of the force and moment vector, $[X\ Y\ Z\ K\ M\ N]^T$, are shown in the RUV coordinate frame. 14
Figure 3.2. Vehicle symmetry and the drag induced moment. 31
Figure 3.3. Definitions of the body lift nomenclature. 37
Figure 3.4. Definitions of the fin lift attack angles for the y- and z-direction fin lift forces in the vehicle reference frame. Also shown are the respective attack angles and moment arm lfin. 40
Figure 3.5. Inherent open-loop heading instability of the RUV. The graph shows the horizontal plane motion in inertial coordinates of the RUV for a 20 second period. The rudder fins are fixed at -0.5° which, in the absence of the Munk moment, would drive the vehicle in the +y-direction. However, due to the destabilizing heading Munk moment, the RUV actually turns in the direction opposite of its fin command. 52
Figure 3.6. RUV states during open-loop heading maneuver. The plots show the RUV states during the open loop maneuver demonstrating the inherent heading instability of the RUV class. 53
Figure 3.7. Block diagram of the closed loop nonlinear system used for the model verification effort (Equation 3.46). The controller block is comprised of decoupled heading and depth controllers. The surge speed is set in open loop mode by fixing the thruster rpm. 54
Figure 3.8 Depth controller utilizes an inner loop pitch PID and an outer loop depth proportional controller. The response of the pitch loop (dashed box) is required to be sufficiently higher than the outer loop response in order to assume $\theta_d = \theta$ in the control law derivation. 60
Figure 3.9 RUV heading controller utilizing a PD control law. 65
Figure 3.10. Simulated and measured depth (top) and heading (bottom) response of the REMUS RUV. The desired response is shown as the green data series. The control loop rate is nominally 5 Hz. 70
Figure 3.11. REMUS Earth-frame position for the linear control model verification. Simulated (blue) and measured (green) position of the REMUS RUV in meters over a period of approximately 3 minutes (top) and fifteen minutes (bottom). 71

Figure 3.12. REMUS RUV simulation: state response (blue) to the depth and heading input commands shown in Figures 3.6. The experimentally measured responses (green) were available for selected state variables. 72
Figure 4.1. The relation between the globally stable controller, τ_{CT}, and its least squares approximation, τ_{LS}. The error vector, ε, acts as a disturbance on the system since it is perpendicular to the input space spanned by the columns of G, i.e., it is an unmatched disturbance..... 75
Figure 4.2. Regulation performance of the regular form sliding mode control law. 99
Figure 4.3. Tracking performance of the regular form sliding mode control law. 101
Figure 4.4. Robustness performance of the regular form sliding mode control law compared to the PID-type control law of Chapter 3 using the model uncertainty of Equation 4.51. The performance of the linear controllers when no model uncertainty exists is shown in Figure 3.10. 103
Figure 5.1. Image of the experimental RUV. The RUV was developed to test the hypothesis that the regular form sliding mode control law could be implemented in an embedded environment and that the controller would outperform the conventional PID approach in terms of robustness to model uncertainty. 107
Figure 5.2 The RUV servo and thruster assembly. This wetted portion of the RUV hull, shown with fins removed, houses the water-proof fin servos, fin linkage and the single thruster motor. 108
Figure 5.3 Thruster, shaft and propeller assembly shown with fins removed. The thruster utilizes an off-the-shelf bilge pump motor and hobby-grade propeller. 109
Figure 5.4. The RUV electronics package consists of a two bus-connected microcontrollers, sensors, motor controllers, and power conversion/management circuitry. 111
Figure 5.5. Experimental RUV closed loop heading response using the regular form sliding mode control with Model A parameters. The response in heading and yaw rate for three different values of the boundary layer control parameter are shown (δ in Equation 4.15). 120
Figure 5.6. Experimental RUV closed loop heading response using the Chapter 4 sliding mode control with Model B parameters. The response in heading and yaw rate for three different values of the boundary layer control parameter are shown (δ in Equation 4.15). 121
Figure 5.7. Experimental RUV closed loop heading response using the Chapter 3 proportional-derivative heading controller. Responses for both models are shown. The best error response is 5.99 steady-state heading rms for Model B. 122
Figure 6.1. Illustration of the nonstationary nature of uncertainty bounds. A noisy system state (blue) has uncertainty bounds (red) that vary in time (not necessarily symmetric). A typical sliding mode control design bases the selection of the performance term gain on a conservative constant estimate of the maximum dynamic uncertainty (green). 124
Figure 6.2. The Particle Filter state and gain observer timing diagram. The prediction, update, resample and prediction cycle for N=8 particles is shown. The variable k represents the discrete time index. The notation $E[\ ]$ and $Cov[\ ]$ represents the

expectation and covariance operation, respectively. The remaining notation on the right hand side of the diagram is explained in the text. 136
Figure 6.3. Comparison of the Particle Filter uncertainty state and bounds observer (Table 6.2) and EKF (Table 6.1) performance for the system in Equation 5.13. The number of particles is 500 for the PF. For both algorithms, the initial state is x0 ~ N(0,2), and the measurement and process noise are vk ~ N(0, 1) and wk ~ N(0,10), respectively. The figure shows the actual uncertainty with the uncertainty bounds estimates from both the Particle Filter and Extended Kalman Filter techniques.... 138
Figure 6.4. Comparison of the PF predictor (Table 6.2) and EKF (Table 6.1) performance for the system in Equation 6.13. In this figure, the actual and estimated state are shown along with the 2-sigma uncertainty bounds estimates..... 139
Figure 6.5. Example of a sampled data function representation for use with the PF. The data represent samples of the uncertain process dynamics gathered off-line through experimental procedures. The model, in this case represented as a simple table of data with columns as states at time k and rows as states at time k+1, can be used with the PF state and gain observer to appropriately fuse uncertain process dynamics (sampled data) with noisy measurements to obtain model uncertainty bounds..... 142
Figure 6.6. Actual uncertainty bounds for the EKF and PF functional uncertainty estimators. The added functional uncertainty in the form of a sampled system accounts for the higher uncertainty bound estimate. 144
Figure 6.7. The SMCP appropriately fuses sampled process dynamics with noisy measurements to obtain model uncertainty bounds. 145
Figure 6.8. Block diagram of the closed-loop RUV with the Particle Filter (PF) optimally-robust control implementation. Noisy sensor data, y, and the control input, u, are used by the Particle Filter to estimate the RUV state and model uncertainty, x and P_f, respectively. These variables are then used in the sliding mode controller to generate the control signal, u. The control and disturbance, , excite the RUV model (dashed box). 147
Figure 6.9. Heading oscillation caused by the RUV attempting to maintain a zero sway speed. 150
Figure. 6.10. The simplified wave disturbance. Earth- and body-fixed RUV coordinates are shown along with the heading-subsystem state variables as well as each state variable's positive direction. 152
Figure. 6.11. RUV performance baseline. The wave disturbance is turned off in the simulation. The plot shows the heading subsystem state variables. 156
Figure. 6.12. RUV performance baseline: heading subsystem modeling uncertainty when no wave disturbance is present. 157
Figure. 6.13. The EKF state estimation innovations sequence: zero disturbance case.. 158
Figure. 6.14. RUV heading subsystem performance for the EKF state estimate and constant gain case. 159
Figure. 6.15. RUV performance for the EKF state estimate and *constant* gain case: heading subsystem modeling uncertainty. 160
Figure. 6.16. The EKF innovations sequence: constant gain case. 161
Figure. 6.17. RUV heading subsystem performance for the EKF state and gain estimate. 162

Figure. 6.18. RUV heading subsystem performance for the EKF state and gain estimate: heading subsystem modeling uncertainty. 163
Figure. 6.19. The EKF innovations sequence: EKF state and gain estimation case. 164
Figure. 6.20. RUV heading subsystem performance for the PF state and gain observer using 10 particles. 165
Figure. 6.21. RUV heading subsystem performance for the PF state and gain estimate with 10 particles: heading subsystem modeling uncertainty. 166
Figure. 6.22. The PF innovations sequence: PF state and gain estimation case with N=10 particles. 167
Figure. 6.23. RUV heading subsystem performance for the PF state and gain estimate using 20 particles. 168
Figure. 6.24. RUV heading subsystem performance for the PF state and gain estimate with 20 particles: heading subsystem modeling uncertainty. 169
Figure. 6.25. RUV heading subsystem performance for the PF state and gain estimate using 30 particles. 170
Figure. 6.26. RUV heading subsystem performance for the PF state and gain estimate with 30 particles: heading subsystem modeling uncertainty. 171
Figure A.1. RUV schematic showing sensor, fin servo and inter-microcontroller connections. 180
Figure A.2. RUV custom circuit card schematic. 181

Chapter 1: Introduction

The subject of this dissertation is the optimally-robust nonlinear control of a class of robotic underwater vehicles (RUVs) characterized by high fineness ratio (length/diameter), axial symmetry, and passive roll stability. RUVs find application in many areas of scientific and military research as well as offshore commercial ventures such as oil and gas exploration that require long term underwater autonomous task execution [Mindell and Bingham], [Fernandez, et. al.], [Allen et. al. 1997]. This type of vehicle (Figure 1.1) is optimized for underwater robotic applications requiring power efficiency for long-range autonomous operations and motion stability for sensor performance enhancement.

The particular class of RUV is further identified by its inconsistent actuator configuration where the number of inputs is less than the number of degrees of freedom and by the loss of controllability at low surge speeds due to the use of fin-based control surface actuation. Therefore, this important type of RUV constitutes an interesting and challenging class of systems to study from a control theoretic perspective, one that has not been generically treated previously.

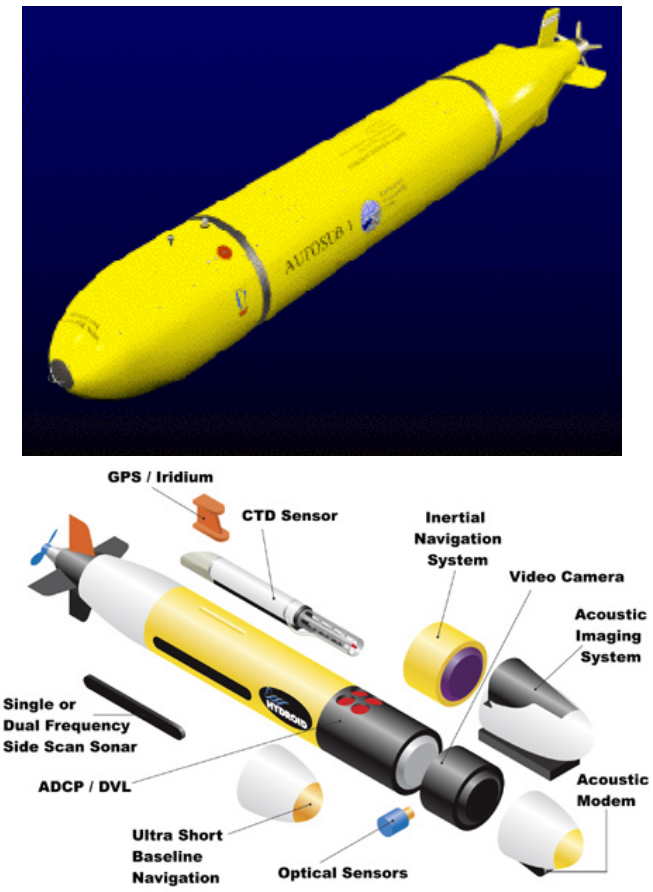

Figure 1.1. Examples of the RUV class described in the text. Top: The University of Southampton's Autosub. Bottom: Woods Hole Oceanographic Institute's modular RUV, the REMUS 100. The illustration demonstrates the REMUS's configuration modularity.

RUV control architectures are typically hierarchical in nature, stacking upper level higher reasoning functionality upon lower level navigation and control algorithms. This lower level is in turn usually segregated into guidance and autopilot segments. The

former translates higher level navigational waypoints into physical state variable setpoints which the autopilot incorporates into its error signal used for control purposes.

Due to several complications, the autopilot control algorithms require special design attention in order to ensure stable, reliable performance over the desired operating range. Complications include the highly nonlinear nature of underwater vehicle dynamics and vehicle designs with inconsistent actuator configurations. The class of vehicles with inconsistent actuator configurations is fully actuated via a set of forces and moments generated through the *dependent* interaction of thrusters and control surfaces with limited angular range and small relative surface area. In the inconsistent system the number of inputs is fewer than the number of degrees of dynamic freedom. Therefore the control signal cannot generically be determined by inverting the input matrix (assuming the system is affine in control). Thus, *computed torque* methods which guarantee global stability in straightforward manner do not apply.

Researchers have therefore turned to methods which assume the surge, depth and heading subsystems can be decoupled. A separate controller is then designed for each decoupled subsystem [Allen et. al. 1997], [Fossen]. An approach which does not assume subsystem separability is Single-input, Multiple State (SIMS) which utilizes a variable structure control approach wherein the vehicle dynamics are linearized about an operating point and a performance control term is injected which cancels the remaining nonlinearities [Cristi et. al.]. However, these methods do not provide any guarantee of stability robustness for the closed loop nonlinear dynamics when modeling uncertainties exist. To the author's knowledge there does not exist a development of the control law design of the RUV class described above when poorly modeled disturbances exist. Such

disturbances include ignored subsystem coupling terms and environmental perturbations. Conventional control techniques for the RUV class of interest include heading and depth PID autopilots [Allen et. al. 1997] and lead compensation [Roup and Humphreys]. Such approaches linearized around a nominal operating condition in order to derive control gains. Such an approach is justified in situations where the RUV mission is well defined and the requirement for autonomy is less emphasized. However, present and future applications of RUVs necessitate greater flexibility and autonomy which in turn place higher demands on the controller for robustness and optimality over a wider operating regime.

The methods of this dissertation attempt to broaden the region of tracking stability of RUV control systems in the presence of poorly modeled environmental disturbances by providing a comprehensive and generic approach to the optimally-robust control design for the class of robotic underwater vehicles (RUVs) characterized by the attributes mentioned. For this purpose a robust nonlinear control technique is required. In this dissertation regular form sliding mode control is developed for the RUV class since the approach allows one to reject modeling uncertainty and disturbances acting in the range of the input distribution (in linear systems theory this is analogous to the linear space spanned by the columns of the input matrix [Strang]), thereby allowing the system to behave more robustly.

The enhanced robustness provided by nonlinear sliding mode control comes at the price of increased control activity which is proportional to the disturbance forces and the (bounded) modeling uncertainty magnitudes. Therefore, in order to minimize the required control energy and thereby optimize the robust control, the performance term

gain and the sliding manifold coefficients can be adjusted according to an estimate of the modeling uncertainty bounds. A novel approach is developed which utilizes Particle Filter techniques to estimate the RUV state and modeling/disturbance uncertainty. The uncertainty estimate is used to dynamically adjust the sliding mode controller response. As a result, the RUV experiences increased performance in terms of heading stability and decreased actuator power consumption.

DISSERTATION CONTRIBUTION

The primary contribution of this dissertation is the formulation and solution of the optimally-robust control problem for the class of robotic underwater vehicles characterized generically by inherent heading instability, high fineness ratio, axial symmetry, metacenter stability, and an inconsistent actuator configuration. The approach involves showing that a regular form sliding mode control law can be designed for the heading and depth subsystems which will provide RUV tracking and regulation stability and that the controller gains and sliding manifold coefficients can be chosen in such a way as to simultaneously provide robust tracking stability and satisfy certain optimality conditions in the presence of disturbances and modeling uncertainty. As will be demonstrated, the technique involves estimating the time varying modeling uncertainty of a nonlinear plant using a nonlinear one step ahead state estimation approach utilizing Particle Filtering methods. Originally [Buckner, Fernandez, Masada] and [Fernandez and Buckner] presented an offline regression method for estimating the confidence interval related to modeling uncertainties for use in the optimal sliding mode control gain using

radial basis neural networks. However, this technique assumed that the full state is accurately measurable during online implementation. Problems occur when only noisy state measurements are available at run time because such techniques cannot optimally fuse model estimates with noisy measurements. The methods of this dissertation address this issue.

Thus, a second contribution of the present thesis is the formulation and solution of a generic approach using Particle Filter techniques to the estimation of nonlinear plant modeling uncertainty bounds and their use in minimizing a performance index relating the state error and control energy to the sliding surface coefficients and the performance control gain. This combined approach yields an adaptive optimally-robust nonlinear controller.

A further contribution of this dissertation is the application of the foregoing nonlinear control methodology to the tracking control of a simulated instantiation of a vehicle from the aforementioned class of RUVs (viz., Hydroid's REMUS [Hydroid]). To the author's knowledge this presents the first instance of such an application since the canonical RUV control for the REMUS utilizes linear PID methods [Allen et. al. 1997]. The regular form sliding mode control technique is also applied to an actual RUV of the class described. The resulting data show that the presented methods outperform a conventional heading proportional-derivative autopilot in terms of the heading steady-state error and robustness to model variation.

A final offering of this dissertation is the rigorous development and verification of a general 6 degree of freedom mathematical model of a closed loop RUV autopilot algorithm for the specified class of RUV. The verification analysis is performed for the

REMUS 100 [Hydroid] shown in Figure 1. Previous works have focused only on the development and qualitative verification of open loop models [Prestero].

DISSERTATION ORGANIZATION

The present dissertation is organized as follows. Chapter 2 reviews the relevant literature. Chapter 3 presents the derivation of the robot dynamic model, including a detailed examination of the forces and moments acting on the vehicle. A closed loop model verification stage is performed. The control laws are based on a simple state linearized PID-type controller derived using the assumption that the depth and heading substates are uncoupled or loosely coupled. A comparison is then made between the simulation and experimental closed loop vehicle state data. Chapter 4 presents the development and application of the regular form sliding mode controller to the depth and heading RUV control problem. Performance is analyzed and comparison to the linearized controller of Chapter 3 is made for the case of simulated parameter uncertainty. Chapter 5 presents the development of an experimental RUV and the results of implementing and comparing both the nonlinear regular form sliding mode controller and the conventional proportional-derivative heading controller. Chapter 6 motivates the need for nonlinear sequential optimal estimation techniques in order to optimize the regular form sliding mode control law performance term. A method is developed for predicting the model uncertainty of a nonlinear dynamic system when both model and state uncertainty exists. The approach is demonstrated using an example system with hard nonlinearities and compared to both a constant gain and an Extended Kalman state

and gain observer algorithm. Finally, Chapter 7 reviews the key dissertation accomplishments and lists several recommendations for further research.

CHAPTER 2: LITERATURE REVIEW

INTRODUCTION

This section briefly reviews the relevant literature that has provided the general background material and guided the innovations detailed within this dissertation proposal.

RUV MODEL

The general 6 degree of freedom motion of a Navy class submarine was ultimately standardized in [Feldman] which used the standardized notation of [SNAME]. The varied use of underwater robotic vehicles has inspired many designs encompassing body type and thruster configuration. [Yoerger and Slotine] investigated the Experimental Autonomous Vehicle (EAVE) and a model of the ROV Jason is developed in [Yoerger, et. al.]. The REMUS robotic vehicle was chosen for the investigation of this dissertation. It is a closed hull, single thruster torpedo-shaped vehicle actuated by separate stern and rudder control surfaces. [Prestero] developed an early model for the REMUS and presented a qualitative model verification with open loop performance data from a real vehicle. The treatment of [Fossen] develops the equations of motion and force and moment terms for the general case of 6 degree of freedom motion of a body submerged in a fluid and simplified model equations for illustrative purposes, but focuses

mainly on surface vessels. An investigation into underwater vehicle body design for optimized hydrodynamic performance can be found in [Paster].

NONLINEAR SLIDING MODE CONTROL

General variable structure control

Sliding mode control is a variable structure, robust control technique utilizing the superposition of two control signals [Utkin], [Slotine and Li], [Isidori]. The first is a performance term which guarantees that system will reach a designer defined error surface in finite time in the presence of noise and modeling uncertainties. The error surface, a submanifold of the state space, is a function of the state variables. The second term is designed to drive the error dynamics (the dynamics of the system restricted to the error surface) to zero. A recommended tutorial can be found in [DeCarlo, et. al.]. As with any technique, certain restrictions apply to the class of applicable systems. Such restrictions are related to allowable transformations on the system state variables. The so-called normal form is discussed in [Slotine and Li] and the less restrictive regular form applied to nonlinear systems is discussed in [Perruquetti, et. al.]. [Lu and Spurgeon] investigate the approach for linear systems.

Robotic Underwater Vehicle Control

Basic methods of underwater vehicle control are discussed in [Fossen]. Simple examples of linear control techniques are elucidated, along with more generic approaches

when the number of control actuators is larger than the number of degrees of freedom. Several stability results are proven which assume an invertible input matrix (the number of control actuators is greater than the number of degrees of freedom). The results are not applicable, however, since the number of degrees of freedom are greater than the number of control actuators for the robot class discussed in this dissertation. Feedback linearization under the invertible input matrix assumption is addressed in [Fossen] as well. [Cristi, et. al.] develops a hybrid state linearized/sliding mode control approach. The single-input multiple states (SIMS) procedure utilizes a hybrid sliding mode control method where the vehicle dynamics are linearized about an operating point and a performance control term is injected which cancels the remaining nonlinearities. A second control signal is injected that asserts the linear feedback performance. The gain matrix of the second term can be computed using pole placement while the sliding surface coefficients are computable from a matrix eigenvalue equation in the dual space of the linearized dynamics. Finally, the underwater robotic control laws derived in references [Yoerger and Slotine] and [Yoerger, et. al.] assume the more restrictive normal form for the equations of motion in order to facilitate the calculation of the sliding mode surface coefficients.

OPTIMAL GAIN SELECTION AND NONLINEAR STATE ESTIMATION

Adaptive gain sliding mode control laws that use artificial intelligence methods for bounding modeling uncertainty have been developed in [Fernandez and Buckner] and [Buckner, Fernandez and Masada]. The techniques utilize radial basis neural nets

(RBNNs) to approximate modeling uncertainty as a function of the system state. The method involves appropriately biasing the learning error of the RBNN such that the data variance is estimated instead of the mean. The approach assumes that the state is accurately measurable or available from a deterministic observer during on-line implementation. The model uncertainty bound is then used in the sliding mode control performance term in order to minimize the required control energy.

In this dissertation a novel stochastic observer for estimating the system state and control gain for the robust control law will be developed. It is based on Particle Filter techniques which offer an approximate solution to the sequential Bayes estimation problem. The basic algorithm was first demonstrated in [Gordon, et. al.]. An instructive reference tutorial is found in [Arulampalam, et. al.]. Some implementation issues with particle filtering can be found in the reference [Daum and Huang] which bounds the computational complexity of the algorithm as a function of the number of particles and the state space dimension. The text [Doucet, et. al.] discusses recent advances and applications of particle filters.

Finally, in uncertain robust control methods an issue arises concerning the combined observer-controller stability for nonlinear closed-loop systems. Since no general separation principle exists for nonlinear control-observer systems, each problem must be considered separately. A method for a class of surface ships is developed in [Loria, et. al.], whereas [Cristi, et. al.] discuss the separation principle in the context of the aforementioned SIMS control approach for underwater vehicles.

Chapter 3: Mathematical Model of the RUV Class

VEHICLE DESCRIPTION

The class of RUVs studied in this dissertation is characterized by a high fineness ratio (body length/ maximum diameter), axial symmetry, and metacenter stability (passively stable in pitch and roll). These RUVs are typically less than 20 inches in diameter with fineness ratios of 6-11. [Paster] provides a detailed characterization of the hull design for the present RUV class that optimizes hydrodynamic performance. With the aim of this research being to develop optimally-robust RUV controls which are model based, it is imperative to develop a good mathematical model. However, the modeling process is complicated due to the highly nonlinear nature of underwater vehicle dynamics. The goal of this chapter is to develop and verify a mathematical description of the RUV dynamics that provides a reasonable level of fidelity adequate for control law development.

COORDINATE SYSTEM AND STATE VARIABLES

Figure 3.1 shows a schematic of the relation between the Earth-fixed and body-fixed coordinate systems used in the RUV modeling equations where the notation follows the SNAME convention [SNAME].

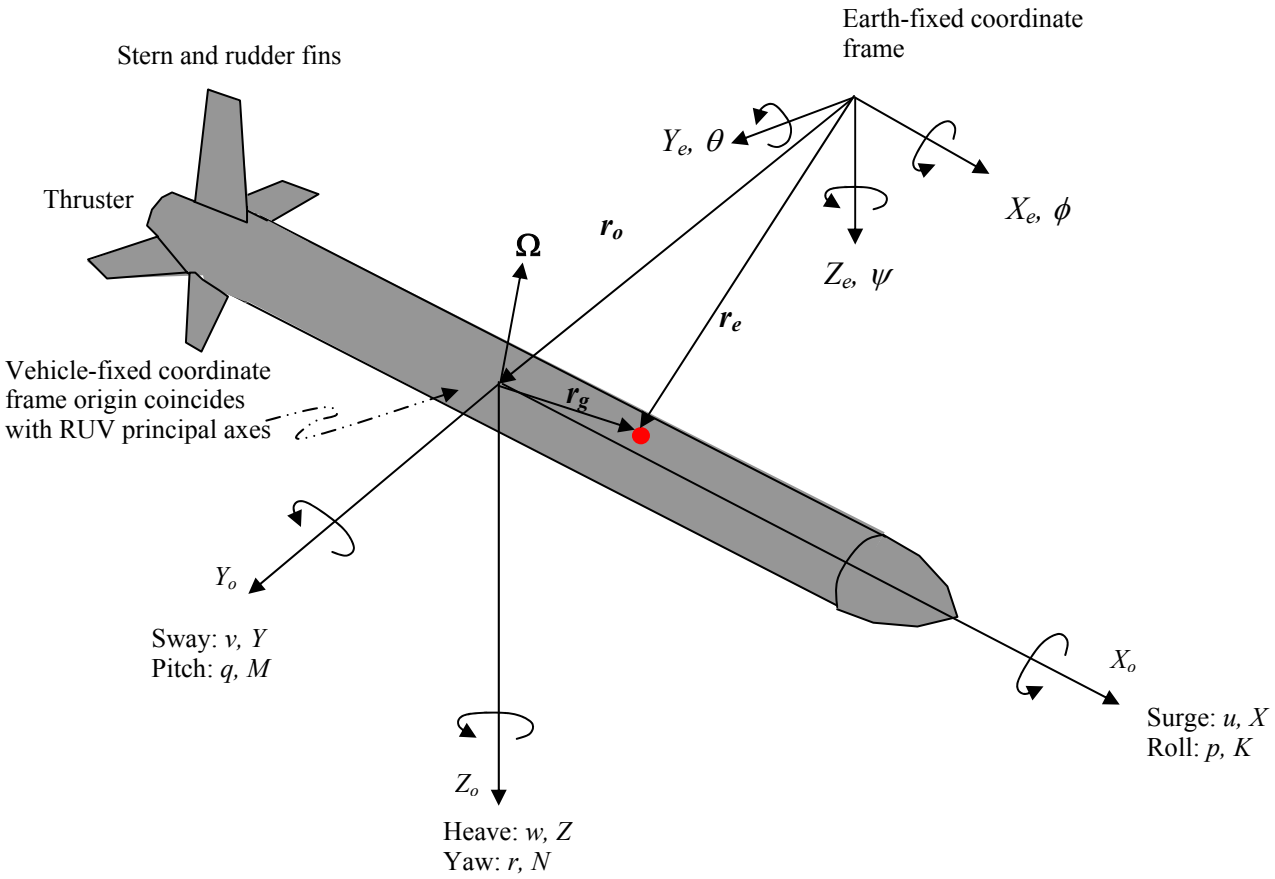

Figure 3.1. Earth-fixed and body-fixed coordinate systems. The RUV center of gravity is shown as a red dot. The spinning arrows indicate sense of rotation of the Euler angles ($\varphi,\ \theta,\ \psi$) and the body-fixed angular momentum vector $\Omega = (p,\ q,\ r)$. The linear velocity vector of the RUV is ($u,\ v,\ w$). The location of the RUV reference frame origin with respect to the Earth-fixed reference frame origin is r_o. The location of the RUV center of gravity with respect to the Earth-fixed reference frame origin is r_e. The RUV center of gravity is fixed at r_g with respect to the RUV frame. Finally, the components of the force and moment vector, $[X\ Y\ Z\ K\ M\ N]^T$, are shown in the RUV coordinate frame.

Figure 3.1 shows the RUV center of gravity along with the sense of rotation of the Euler angles (φ, θ, ψ) and their body-fixed counterpart momenta (p, q, r). The RUV coordinate system is chosen to coincide with the principal axes of inertia in order to justify later approximations in the equations of motion which depend on small relative magnitudes of products of inertia. The system state variable vectors, ν and η, and the force and moment vector, τ, are defined, respectively, as

$$\begin{aligned} \nu &= \begin{bmatrix} \nu_1^T & \nu_2^T \end{bmatrix}^T \quad \nu_1 = \begin{bmatrix} u & v & w \end{bmatrix}^T \quad \nu_2 = \begin{bmatrix} p & q & r \end{bmatrix}^T \\ \eta &= \begin{bmatrix} \eta_1^T & \eta_2^T \end{bmatrix}^T \quad \eta_1 = \begin{bmatrix} x & y & z \end{bmatrix}^T \quad \eta_2 = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T \quad . \\ \tau &= \begin{bmatrix} \tau_1^T & \tau_1^T \end{bmatrix}^T \quad \tau_1 = \begin{bmatrix} X & Y & Z \end{bmatrix}^T \quad \tau_2 = \begin{bmatrix} K & M & N \end{bmatrix}^T \end{aligned} \tag{3.1}$$

COORDINATE TRANSFORMATION MATRIX AND VEHICLE KINEMATICS

Rigid body kinematics are essentially linear and angular velocity transformations between coordinate systems. According to the Euler Rotation Theorem [Fossen], every change in the relative orientation of two rigid bodies can be produced by a simple sequence of rotations of the RUV reference system within the Earth-fixed coordinate system. The order of rotations is not arbitrary. The xyz convention of Euler angle rotations is heretofore adopted. The convention is to translate the Earth-fixed frame parallel to itself until both origins coincide and then rotate in heading angle, followed by pitch and then roll angle rotations to obtain the body-fixed coordinate system. See

[Fossen] for details. The result is that the Earth-fixed linear and angular velocities are related to the body fixed velocities by the equation

$$\dot{\eta} = J(\eta) \cdot \nu \tag{3.2}$$

where

$$J(\eta) = \begin{bmatrix} J_1(\eta_2) & 0_{3x3} \\ 0_{3x3} & J_2(\eta_2) \end{bmatrix} \tag{3.3}$$

and

$$J_1(\eta_2) \equiv \begin{bmatrix} \cos\psi\cos\theta & -\sin\psi\cos\phi + \cos\psi\sin\theta\sin\phi & \sin\psi\sin\phi + \cos\psi\cos\phi\sin\theta \\ \sin\psi\cos\theta & \cos\psi\cos\phi + \sin\phi\sin\theta\sin\psi & -\cos\psi\sin\phi + \sin\theta\sin\psi\cos\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix} \tag{3.4}$$

$$J_2(\eta_2) = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta} \end{bmatrix}. \tag{3.5}$$

Note that since $J_1(\eta_2)$ describes a coordinate transformation matrix it is orthogonal. Therefore $J_1^{-1}(\eta_2) = J_1^T(\eta_2)$.

DERIVATION OF THE COMPLETE RUV EQUATIONS OF MOTION

In order to derive the RUV equations of motion in the vehicle reference frame one must use the following result from rigid body mechanics that relates the time derivative of a vector quantity, Q, in Earth-fixed and RUV-fixed reference frames [Marion and Thornton]

$$\overset{\circ}{Q} \equiv \left(\frac{dQ}{dt} \right)_{Earth} = \dot{Q} + \Omega \times Q \tag{3.6}$$

where Ω is the body fixed coordinate frame angular velocity vector (also the angular velocity vector of the RUV since the rotating coordinate frame is attached to the RUV), $\overset{\circ}{Q}$ denotes a time derivative in the inertial frame, and $\dot{Q}$ denotes a time derivative in the non-inertial frame. Note that this implies

$$\overset{\circ}{\Omega} = \dot{\Omega} + \Omega \times \Omega = \dot{\Omega} \tag{3.7}$$

so that the angular acceleration vector is equivalent in both Earth-fixed and RUV-fixed frames. Now let the quantity Q be the vehicle's linear velocity, temporarily denoted as U. Newton's equations of motion, valid only in the inertial or Earth-fixed frame, are given by

$$\begin{aligned} \sum F_{ext} &= m \overset{\circ}{U} \\ \sum \tau_{ext} &= I \cdot \overset{\circ}{\Omega} + r_g \times m \overset{\circ\circ}{r_o} \end{aligned} \tag{3.8}$$

where I is the time invariant RUV inertia tensor referenced to the origin. The second term on the right hand side of the equation for $\sum \tau_{ext}$ is included since the RUV origin is not coincident with the center of gravity of the vehicle resulting in non-centroidal

rotation. That is, the sum of the external forces acting at the center of gravity creates an additional moment about the RUV coordinate frame origin. From Figure 3.1 the position vectors satisfy $r_e = r_o + r_g$. Therefore

$$\sum F_{ext} = m\overset{\circ}{U} = m\overset{\circ\circ}{r_e} = m\left(\overset{\circ\circ}{r_o} + \overset{\circ\circ}{r_g}\right).$$

Noting that $\overset{\circ\circ}{r_o} \equiv \overset{\circ}{U}_o$ and $\dot{r}_g = 0$ (not always obvious since some vehicle designs affect depth and heading control with internal sliding weights), and using Equations (3.6) and (3.7) one obtains

$$\sum F_{ext} = m\left(\overset{\circ}{U}_o + \overbrace{\left(\dot{r}_g + \Omega \times r_g\right)}^{\circ}\right) = m\left(\dot{U}_o + \Omega \times U_o + \dot{\Omega} \times r_g + \Omega \times (\Omega \times r_g)\right) \qquad (3.9)$$

.

Thus the external force is the sum of linear, Coriolis, angular and centripetal acceleration terms, respectively. Following the same development, the moment equation becomes

$$\sum \tau_{ext} = \overbrace{I \cdot \Omega}^{\circ} + r_g \times m\overset{\circ\circ}{r_o} = I \cdot \dot{\Omega} + \dot{\Omega} \times I \cdot \Omega + r_g \times m(\dot{U}_o + \Omega \times U_o) \qquad (3.10)$$

.

The 6 degree of freedom rigid body equations of motion are therefore

$$
\begin{aligned}
&m(\dot{U}_o + \Omega \times U_o + \dot{\Omega} \times r_g + \Omega \times \Omega \times r_g) = \sum F_{ext} \\
&m r_g \times (\dot{U}_o + \Omega \times U_o) + I \cdot \dot{\Omega} + \dot{\Omega} \times I \cdot \Omega = \sum \tau_{ext}
\end{aligned}
\tag{3.11}
$$

In what follows, let $U_o = \nu_1$ and $\Omega = \nu_2$. Also let the *xyz* unit vectors in the RUV frame be $\hat{i}$, $\hat{j}$ and $\hat{k}$, respectively. Beginning with Equation 3.10, one has

$$
I \cdot \dot{\Omega} = \begin{bmatrix} I_x \dot{p} & -I_{xy} \dot{q} & -I_{xz} \dot{r} \\ -I_{xy} \dot{p} & I_y \dot{q} & -I_{yz} \dot{r} \\ -I_{xz} \dot{p} & -I_{yz} \dot{q} & I_z \dot{r} \end{bmatrix}.
$$

Also,

$$
\begin{aligned}
\dot{\Omega} \times I \cdot \Omega = {} & \hat{i}\left[q\left(-I_{xz} p - I_{yz} q + I_z r\right) - r\left(-I_{xy} p + I_y q - I_{yz} r\right)\right] \\
& + \hat{j}\left[r\left(I_x p - I_{xy} q - I_{xz} r\right) - p\left(-I_{xz} p - I_{yz} q + I_z r\right)\right] \\
& + \hat{k}\left[p\left(-I_{xy} p + I_y q - I_{yz} r\right) - q\left(I_x p - I_{xy} q - I_{xz} r\right)\right] \\
= {} & \hat{i}\left[I_z qr - I_{xz} pq - I_{yz} q^2 + I_{yz} r^2 + I_{xy} rp - I_y qr\right] \\
& + \hat{j}\left[I_x pr - I_{xy} qr - I_{xz} r^2 + I_{xz} p^2 + I_{yz} qp - I_z rp\right] \\
& + \hat{k}\left[I_y pq - I_{xy} p^2 - I_{yz} rp - I_x pq - I_{xy} q^2 - I_{xz} rq\right].
\end{aligned}
$$

Component-wise after expanding and collecting terms this becomes

$$
\begin{aligned}
\hat{i} \cdot \left[I \cdot \Omega + \Omega \times I\Omega \right] &= I_x \dot{p} + I_{xy}\left(pr - \dot{q}\right) - I_{xz}\left(pq + \dot{r}\right) + I_{yz}\left(r^2 - q^2\right) + \left(I_z - I_y\right) rq \\
\hat{j} \cdot \left[I \cdot \Omega + \Omega \times I\Omega \right] &= I_y \dot{q} - I_{xy}\left(qr + \dot{p}\right) + I_{yz}\left(pq - \dot{r}\right) - I_{xz}\left(r^2 - p^2\right) + \left(I_x - I_z\right) pr \\
\hat{k} \cdot \left[I \cdot \Omega + \Omega \times I\Omega \right] &= I_z \dot{r} - I_{xz}\left(qr + \dot{p}\right) - I_{yz}\left(rp + \dot{q}\right) - I_{xy}\left(p^2 + q^2\right) - \left(I_x - I_y\right) pq
\end{aligned}.
$$

The remaining term in Equation 3.10 is

$$\begin{aligned} r_g \times m\left(\dot{U}_o + \Omega \times U_o\right) &= r_g \times m\left(\hat{i}\left(\dot{u} + qw - rv\right) + \hat{j}\left(\dot{v} + ur - pw\right) + \hat{k}\left(\dot{w} + vp - qu\right)\right) \\ &= \hat{i}m\left(y_g\left(\dot{w} + vp - qu\right) - z_g\left(\dot{v} + ur - pw\right)\right) \\ &+ \hat{j}m\left(z_g\left(\dot{u} + qw - rv\right) - x_g\left(\dot{w} + vp - qu\right)\right) \\ &+ \hat{k}m\left(x_g\left(\dot{v} + ur - pw\right) - y_g\left(\dot{u} + qw - rv\right)\right). \end{aligned}$$

Looking at the force equation (Equation 3.9) one has

$$\dot{\Omega} \times r_g = \hat{i}\left(z_g\dot{q} - y_g\dot{r}\right) + \hat{j}\left(x_g\dot{r} - z_g\dot{p}\right) + \hat{k}\left(y_g\dot{p} - x_g\dot{q}\right)$$

and

$$\Omega \times (\Omega \times r_g) = \hat{i}\left(q\left(y_g p - x_g q\right) - r\left(x_g r - z_g p\right)\right) + \hat{j}\left(r\left(z_g q - y_g r\right) - p\left(y_g p - x_g q\right)\right) + \hat{k}\left(p\left(x_g r - z_g p\right) - q\left(z_g q - y_g r\right)\right)$$

The remaining term expands as

$$\Omega \times U_o = \hat{i}\left(qw - rv\right) + \hat{j}\left(ur - pw\right) + \hat{k}\left(vp - qu\right).$$

Combining these equations yields the components

$$\hat{i}: \quad \dot{u} - x_g\left(r^2 + q^2\right) + y_g\left(-\dot{r} + qp\right) + z_g\left(\dot{q} + rp\right) + qw - rv$$
$$\hat{j}: \quad \dot{v} + x_g\left(\dot{r} + pq\right) - y_g\left(r^2 + p^2\right) + z_g\left(rq - \dot{p}\right) + ru - pw$$
$$\hat{k}: \quad \dot{w} + x_g\left(rp - \dot{q}\right) + y_g\left(rq + \dot{p}\right) - z_g\left(q^2 + p^2\right) + pv - qu \,.$$

The expanded 6 DOF rigid body equations of motion are thus

$$\begin{aligned}
&m\left(\dot{u} + wq - vr - x_g\left(r^2 + q^2\right) + y_g\left(-\dot{r} + qp\right) + z_g\left(\dot{q} + rp\right)\right) = \sum X_{ext} \\
&m\left(\dot{v} + ru - pw + x_g\left(\dot{r} + pq\right) - y_g\left(r^2 + p^2\right) + z_g\left(rq - \dot{p}\right)\right) = \sum Y_{ext} \\
&m\left(\dot{w} + pv - qu + x_g\left(rp - \dot{q}\right) + y_g\left(rq + \dot{p}\right) - z_g\left(q^2 + p^2\right)\right) = \sum Z_{ext} \\
&I_x\dot{p} + I_{xy}\left(pr - \dot{q}\right) - I_{xz}\left(pq + \dot{r}\right) + I_{yz}\left(r^2 - q^2\right) + \left(I_z - I_y\right)rq + m\left(y_g\left(\dot{w} + vp - qu\right) - z_g\left(\dot{v} + ur - pw\right)\right) = \sum K_{ext} \\
&I_y\dot{q} - I_{xy}\left(qr + \dot{p}\right) + I_{yz}\left(pq - \dot{r}\right) - I_{xz}\left(r^2 - p^2\right) + \left(I_x - I_z\right)pr + m\left(z_g\left(\dot{u} + qw - rv\right) - x_g\left(\dot{w} + vp - qu\right)\right) = \sum M_{ext} \\
&I_z\dot{r} - I_{xz}\left(qr + \dot{p}\right) - I_{yz}\left(rp + \dot{q}\right) - I_{xy}\left(p^2 + q^2\right) - \left(I_x - I_y\right)pq + m\left(x_g\left(\dot{v} + ur - pw\right) - y_g\left(\dot{u} + qw - rv\right)\right) = \sum N_{ext}
\end{aligned} \tag{3.12}$$

EXTERNAL FORCES AND MOMENTS

This section details the derivation of the external forces and moments which comprise the right hand side of Equation 3.12. The external forces and moments are the sum of added mass, hydrodynamic damping, body and fin lift, hydrostatic restoring forces and moments, and propulsion. The [SNAME] notational convention common in marine systems modeling is adopted to symbolize hydrodynamic coefficients. For example, the quadratic hydrodynamic drag force in the x-direction, X_{drag}, is given by

$$X_{drag} = X_{u|u|}u|u|$$

where $X_{u|u|}$ is the force coefficient defined by

$$X_{u|u|} \equiv \frac{\partial X_{drag}}{\partial u|u|}.$$

Added Mass

Added mass, also known as virtual mass, is the additional apparent inertia arising from the entrainment of the surrounding fluid media by the vehicle motion. It is frequently understood to be the set of pressure-induced forces and moments acting on an immersed body by the surrounding fluid undergoing harmonic motion. Consequently, like actual inertial mass, added mass can be separated into inertial, $M_A\dot{\nu}$, and Coriolis/centripetal, $C_A(\nu)\nu$, force and moment contributions where

$$M_A = \begin{bmatrix} X_{\dot{u}} & X_{\dot{v}} & X_{\dot{w}} & X_{\dot{p}} & X_{\dot{q}} & X_{\dot{r}} \\ Y_{\dot{u}} & Y_{\dot{v}} & Y_{\dot{w}} & Y_{\dot{p}} & Y_{\dot{q}} & Y_{\dot{r}} \\ Z_{\dot{u}} & Z_{\dot{v}} & Z_{\dot{w}} & Z_{\dot{p}} & Z_{\dot{q}} & Z_{\dot{r}} \\ K_{\dot{u}} & K_{\dot{v}} & K_{\dot{w}} & K_{\dot{p}} & K_{\dot{q}} & K_{\dot{r}} \\ M_{\dot{u}} & M_{\dot{v}} & M_{\dot{w}} & M_{\dot{p}} & M_{\dot{q}} & M_{\dot{r}} \\ N_{\dot{u}} & N_{\dot{v}} & N_{\dot{w}} & N_{\dot{p}} & N_{\dot{q}} & N_{\dot{r}} \end{bmatrix}.$$

According to [Fossen] one can safely assume M_A is symmetric positive definite. Therefore the following theorem concerning the parameterization of $C_A(\nu)$ in terms of M_A applies.

Theorem 3.1 Let $\nu = [\nu_1^T \ \nu_2^T]^T = \begin{bmatrix} u & v & w & p & q & r \end{bmatrix}^T$ and partition M_A accordingly:

$$M_A = \begin{bmatrix} M_{A11} & M_{A12} \\ M_{A12} & M_{A22} \end{bmatrix} \tag{3.13}$$

The Coriolis/centripetal added mass contribution satisfies the parameterization

$$C_A(\nu) = \begin{bmatrix} 0_{3x3} & -S\left(M_{A11}\nu_1 + M_{A12}\nu_2\right) \\ -S\left(M_{A11}\nu_1 + M_{A12}\nu_2\right) & -S\left(M_{A12}\nu_1 + M_{A22}\nu_2\right) \end{bmatrix} \tag{3.14}$$

where S is the skew symmetric matrix operator satisfying

$$S(\lambda) = \begin{bmatrix} 0 & -\lambda_3 & \lambda_2 \\ \lambda_3 & 0 & -\lambda_1 \\ -\lambda_2 & \lambda_1 & 0 \end{bmatrix} \tag{3.15}$$

for any vector $\lambda = \begin{bmatrix} \lambda_1 & \lambda_2 & \lambda_3 \end{bmatrix}^T$.

Proof The proof can be found in [Fossen].

Expressions for the added mass terms can be derived from consideration of the vehicle geometry. Since the class of vehicles of interest in this thesis exhibit strong top-bottom

and port-starboard symmetry one can justify the following form of the added mass matrix [Yue]:

$$M_A = \begin{bmatrix} X_{\dot{u}} & 0 & 0 & 0 & 0 & 0 \\ 0 & Y_{\dot{v}} & 0 & 0 & 0 & Y_{\dot{r}} \\ 0 & 0 & Z_{\dot{w}} & 0 & Z_{\dot{q}} & 0 \\ 0 & 0 & 0 & K_{\dot{p}} & 0 & 0 \\ 0 & 0 & Z_{\dot{q}} & 0 & M_{\dot{q}} & 0 \\ 0 & Y_{\dot{r}} & 0 & 0 & 0 & N_{\dot{r}} \end{bmatrix}. \tag{3.16}$$

Using this and the results of the theorem shows

$$C_A(\nu) = \begin{bmatrix} 0 & 0 & 0 & 0 & Z_{\dot{w}}w + Z_{\dot{q}}q & -\left(Y_{\dot{v}}v + Y_{\dot{r}}r\right) \\ 0 & 0 & 0 & -\left(Z_{\dot{w}}w + Z_{\dot{q}}q\right) & 0 & X_{\dot{u}}u \\ 0 & 0 & 0 & Y_{\dot{v}}v + Y_{\dot{r}}r & -X_{\dot{u}}u & 0 \\ 0 & \left(Z_{\dot{w}}w + Z_{\dot{q}}q\right) & -\left(Y_{\dot{v}}v + Y_{\dot{r}}r\right) & 0 & \left(Y_{\dot{r}}v + N_{\dot{r}}r\right) & -\left(Z_{\dot{q}}w + M_{\dot{q}}q\right) \\ -\left(Z_{\dot{w}}w + Z_{\dot{q}}q\right) & 0 & X_{\dot{u}}u & -\left(Y_{\dot{r}}v + N_{\dot{r}}r\right) & 0 & K_{\dot{p}}p \\ \left(Y_{\dot{v}}v + Y_{\dot{r}}r\right) & -X_{\dot{u}}u & 0 & \left(Z_{\dot{q}}w + M_{\dot{q}}q\right) & -K_{\dot{p}}p & 0 \end{bmatrix} \tag{3.17}$$

The added mass, $M_A\dot{\nu}$, and Coriolis/centripetal, $C_A(\nu)\nu$, force and moment contributions in component form are summarized below:

$$
\begin{aligned}
X_A &= X_{\dot{u}}\dot{u} + \left(Z_{\dot{w}}w + Z_{\dot{q}}q\right)q - \left(Y_{\dot{v}}v + Y_{\dot{r}}r\right)r \\
Y_A &= Y_{\dot{v}}\dot{v} + Y_{\dot{r}}\dot{r} - \left(Z_{\dot{w}}w + Z_{\dot{q}}q\right)p + X_{\dot{u}}ur \\
Z_A &= Z_{\dot{w}}\dot{w} + Z_{\dot{q}}\dot{q} + \left(Y_{\dot{v}}v + Y_{\dot{r}}r\right)p - X_{\dot{u}}uq \\
K_A &= K_{\dot{p}}\dot{p} + \left(Z_{\dot{w}}w + Z_{\dot{q}}q\right)v - \left(Y_{\dot{v}}v + Y_{\dot{r}}r\right)w + \left(Y_{\dot{r}}v + N_{\dot{r}}r\right)q - \left(Z_{\dot{q}}w + M_{\dot{q}}q\right)r \\
M_A &= Z_{\dot{q}}\dot{w} + M_{\dot{q}}\dot{q} - \left(Z_{\dot{w}}w + Z_{\dot{q}}q\right)u + X_{\dot{u}}uw - \left(Y_{\dot{r}}v + N_{\dot{r}}r\right)p + K_{\dot{p}}pr \\
N_A &= Y_{\dot{r}}\dot{v} + N_{\dot{r}}\dot{r} + \left(Y_{\dot{v}}v + Y_{\dot{r}}r\right)u - X_{\dot{u}}uv + \left(Z_{\dot{q}}w + M_{\dot{q}}q\right)p - K_{\dot{p}}pq
\end{aligned}
\tag{3.18}
$$

[Blevins] gives the following formulas for the axial and roll added mass of a slender cylindrical shape

$$
\begin{aligned}
X_{\dot{u}} &= -(.02)\cdot\tfrac{4}{3}\pi\rho\left(\frac{l}{2}\right)\left(\frac{d}{2}\right)^2 \\
K_{\dot{p}} &= -\tfrac{2}{\pi}\rho R_{fin}^4\left(x_{fin2} - x_{fin1}\right)
\end{aligned}
$$

where l is the vehicle length, d is the cylinder diameter, ρ is the fluid density, x_{fin2} and x_{fin1} are the location of the fin extents in the RUV frame, and R_{fin} is the radius of the fin from the vehicle center line. The added mass per unit length for a small strip of a cylindrical hull is given by [Newman]

$$
M_A = \pi\rho r^2\left(x\right)
$$

where $r(x)$ is the radius of the cylinder at the position x from the RUV origin. For the finned section [Blevins] gives the formula

$$M_{fin} = \pi\rho\left(R_{fin}^2 - r^2(x) + r^4(x)/R_{fin}^2\right)$$

With these formulas and noting the RUV symmetry gives $Z_{\dot{w}} = Y_{\dot{v}}$, $Z_{\dot{v}} = -M_{\dot{w}}$, $Y_{\dot{r}} = N_{\dot{v}}$, and $Z_{\dot{q}} = M_{\dot{w}}$, one can calculate the remaining added mass terms in Equation 3.17:

$$\begin{aligned} Y_{\dot{v}} &= -\int_{x_t}^{x_n} M_A dx - \int_{x_{fin1}}^{x_{fin2}} M_{fin} dx \\ &= -\pi\rho\left(\tfrac{d^2}{4} l + \left(R_{fin}^2 - \tfrac{d}{4}^2 + \left(\tfrac{d}{2}\right)^4 / R_{fin}^2\right) w_{fin}\right) \end{aligned}$$

$$\begin{aligned} M_{\dot{w}} &= -\int_{x_t}^{x_n} x M_A dx - \int_{x_{fin1}}^{x_{fin2}} x M_{fin} dx \\ &= -\pi\rho\left(\tfrac{d^2}{4} \cdot \tfrac{x_n^2 - x_t^2}{2} + \left(R_{fin}^2 - \tfrac{d}{4}^2 + \left(\tfrac{d}{2}\right)^4 / R_{fin}^2\right) \tfrac{x_{fin2}^2 - x_{fin1}^2}{2}\right) \end{aligned}$$

$$\begin{aligned} M_{\dot{q}} &= -\int_{x_t}^{x_n} x^2 M_A dx - \int_{x_{fin1}}^{x_{fin2}} x^2 M_{fin} dx \\ &= -\pi\rho\left(\tfrac{d^2}{4} \cdot \tfrac{x_n^3 - x_t^3}{3} + \left(R_{fin}^2 - \tfrac{d}{4}^2 + \left(\tfrac{d}{2}\right)^4 / R_{fin}^2\right) \tfrac{x_{fin2}^3 - x_{fin1}^3}{3}\right) \end{aligned}$$

Hydrostatic Restoring Forces and Moments

The hydrostatic forces and moments associated with the vehicle's buoyancy and weight must be transformed into the vehicle frame of reference. The gravitational force, or vehicle weight, W, acts through the vehicle's center of gravity. The buoyancy force, $-B$, acts through the vehicle's center of buoyancy. The total hydrostatic force is simply

the sum of these two contributions. Using the Earth-to-RUV frame coordinate transformation, $J_1^T(\eta_2)$, one has

$$\begin{bmatrix} X_{HS} \\ Y_{HS} \\ Z_{HS} \end{bmatrix} = f_g + f_b \tag{3.19}$$

where

$$f_g = J_1^T(\eta_2) \cdot \begin{bmatrix} 0 \\ 0 \\ W \end{bmatrix}, \quad f_b = J_1^T(\eta_2) \cdot \begin{bmatrix} 0 \\ 0 \\ -B \end{bmatrix}.$$

This shows

$$\begin{aligned} X_{HS} &= -(W-B)\sin\theta \\ Y_{HS} &= (W-B)\cos\theta\sin\phi \\ Z_{HS} &= (W-B)\cos\theta\cos\phi \end{aligned} \quad . \tag{3.20}$$

Similarly, the hydrostatic moments are transformed into the RUV frame as

$$\begin{bmatrix} K_{HS} \\ M_{HS} \\ N_{HS} \end{bmatrix} = M_g + M_b \tag{3.21}$$

where

$$M_b = r_b \times f_b = \begin{bmatrix} -y_b B \cdot \cos\theta\cos\phi + z_b B \cdot \cos\theta\sin\phi \\ z_b B \cdot \sin\theta + x_b B \cdot \cos\theta\cos\phi \\ x_b B \cdot \cos\theta\sin\phi - y_b B \cdot \sin\theta \end{bmatrix} \tag{3.22}$$

and

$$M_g = r_g \times f_g = \begin{bmatrix} -y_g W \cdot \cos\theta\cos\phi + z_g W \cdot \cos\theta\sin\phi \\ -z_g W \cdot \sin\theta + x_g W \cdot \cos\theta\cos\phi \\ x_g W \cdot \cos\theta\sin\phi + y_g W \cdot \sin\theta \end{bmatrix}. \tag{3.23}$$

The definitions of r_g and r_b are shown in Figure 3.1. Finally, the moment components are

$$\begin{aligned} K_{HS} &= \left(y_g W - y_b B\right)\cos\theta\cos\phi - \left(z_g W - z_b B\right)\cos\theta\sin\phi \\ M_{HS} &= -\left(z_g W - z_b B\right)\sin\theta - \left(x_g W - x_b B\right)\cos\theta\cos\phi \\ N_{HS} &= \left(x_g W - x_b B\right)\cos\theta\sin\phi + \left(y_g W - y_b B\right)\sin\theta \end{aligned} \quad . \tag{3.24}$$

Hydrodynamic Damping

Hydrodynamic damping forces and moments consist of terms opposing linear and angular motion due to the relative movement of a viscous fluid in contact with the vehicle body. If one assumes for now the RUV operates at a depth sufficient to ignore radiation-induced potential damping from wave action, two primary contributions to the damping components consist of pressure or form drag (function of shape and frontal area) and skin friction or viscous drag (function of speed and wetted surface area) [Yue], [Paster]. Form drag is obtained by integrating the pressure normal to the body surface. This is the

primary component of drag when the body is blunt. Skin drag is obtained by integrating the viscous stresses tangential to the body boundary. The total drag coefficient is related to both the form and skin (friction) drag coefficients, c_{dF} and c_{ds}, respectively, given by [Paster]

$$c_{dF} = \frac{F_{df}}{\frac{1}{2}\rho V^2 A_f}, \quad c_{ds} = \frac{F_{ds}}{\frac{1}{2}\rho V^2 S_w}$$

where A_f is the frontal area, S_w is the total wetted surface area in contact with the fluid, u is the speed of the fluid, ρ is the fluid density, F_{df} and F_{ds} are the form skin drag forces, respectively. To obtain the total drag coefficient one must utilize the ratio of wetted to frontal areas which according to [Paster] can be approximated for high fineness ratio vehicles (length/maximum diameter ≥ 5) by the formula

$$S_w / A_f = 4\left[L/d - 1\right].$$

The total coefficient can then be computed as

$$c_d = c_{ds} + c_{dF} / 4\left[L/d - 1\right] . \tag{3.25}$$

As an example, [Paster] claims c_{dF} can be as low as 0.1 for a fineness ratio of 5 and that c_{ds} is approximately 0.004 at a Reynolds number of $1.5\text{x}10^6$. Using the above formulas, $c_d = 0.0102$. The RUV empirical total drag coefficient in [Prestero], converted using Equation 3.25, is .0112 for a Reynolds number of $1.3\text{x}10^6$ and fineness ratio of 6.96.

For a real vehicle the hydrodynamic drag is a complex, coupled, highly nonlinear function of the Reynolds number and vehicle shape, among other parameters. In order to make the modeling problem tractable the following assumptions are used to shorten the list of required force and moment coefficients:

- Only quadratic drag terms are significant since linear skin friction is only important in the laminar boundary layer [Fossen]. Because a real vehicle will have small imperfections in the surface of its body, the boundary layer is likely to be turbulent where quadratic effects will dominate.
- Linear-angular coupled terms such as X_{wq}, M_{rv}, etc. can be neglected [Fossen].
- The vehicle is top-bottom and port-starboard symmetric. Thus one can neglect terms due to $M_{u|u|}$, etc., (i.e. the drag-induced moments). Figure 3.2 illustrates this assumption for the symmetry of the proposed vehicle class. Vehicle symmetry allows one to ignore the drag induced moment $K_{v|v|}v|v|$ due to sway motion (top diagram). This approximation is used for the drag induced moment $M_{u|u|}u|u|$ due to surge motion (bottom diagram) as well. Finally, though not shown, a similar argument holds for the drag induced moment in heave, $N_{w|w|}w|w|$

- Drag coefficients are functions only of Reynolds number and geometry.

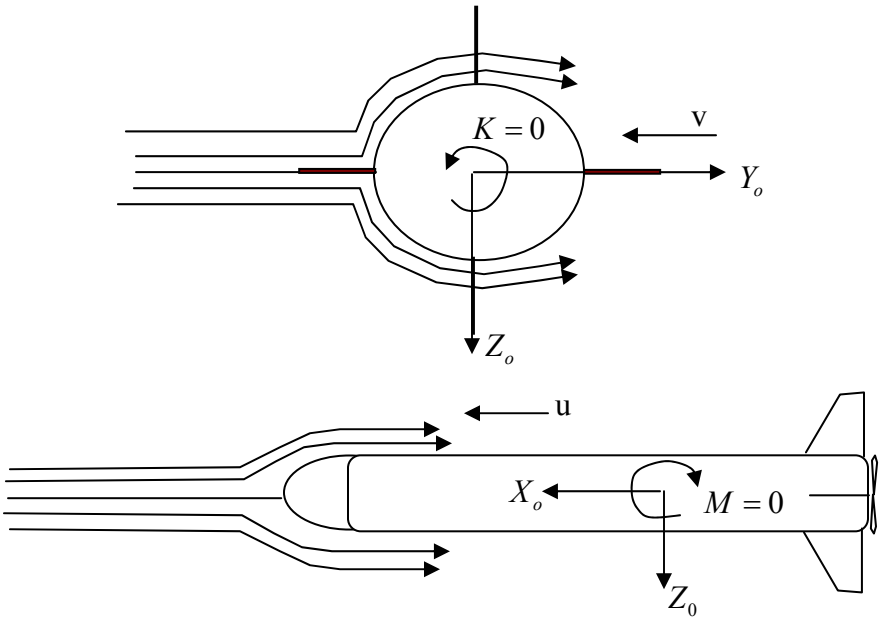

Figure 3.2. Vehicle symmetry and the drag induced moment.

Under the foregoing assumptions the structure of the damping forces and moments vector is $D(\nu)\nu$ where

$$D(\nu) = \begin{bmatrix} X_{u|u|}|u| & 0 & 0 & 0 & 0 & 0 \\ 0 & Y_{v|v|}|v| & 0 & 0 & 0 & Y_{r|r|}|r| \\ 0 & 0 & Z_{w|w|}|w| & 0 & Z_{q|q|}|q| & 0 \\ 0 & 0 & 0 & K_{p|p|}|p| & 0 & 0 \\ 0 & 0 & M_{w|w|}|w| & 0 & M_{q|q|}|q| & 0 \\ 0 & N_{v|v|}|v| & 0 & 0 & 0 & N_{r|r|}|r| \end{bmatrix}. \tag{3.26}$$

The matrix terms are derived next. Since lengthwise the pertinent RUV class exhibits high fineness ratio, its wetted area can be approximated by the planform area in analogy to a wing shape [Fox]. Then the vehicle axial drag coefficient is approximated by

$$X_{u|u|} \simeq -\tfrac{1}{2}\rho c_{da} S_w \tag{3.27}$$

where A_s is the vehicle wetted surface area and c_{da} is the total axial drag coefficient given by Equation 3.25 above.

Next the remaining crossflow and rolling drag terms in the damping vector $D(\nu)\nu$ above are derived. In the following equations the drag coefficient, $c_{d,}$ is understood to be the total drag coefficient acquired by adding the skin and form coefficients using Equation 3.25. The drag induced force is

$$F_d = \int \tfrac{1}{2}\rho c_d u|u| \cdot dS_w \quad . \tag{3.28}$$

The differential area element is given by

$$dS_w = 2\pi R(\xi)d\xi$$

where $R(\xi)$ is the RUV circumferential length at the point ξ in the $X_oY_oZ_o$ coordinate system perpendicular to the fluid flow. The limits of integration are from x_t to x_n, the location of the tail and nose in the vehicle coordinate system, respectively. The skin drag-induced moment is therefore

$$M_d = \int_{x_t}^{x_n} \tfrac{1}{2}\rho c_d u|u| \cdot \xi \cdot 2\pi R(\xi)d\xi \,. \tag{3.29}$$

For rotational movement Equations (3.28) and (3.29) are modified slightly by noting that the linear and angular speeds at the point ξ are related by $u = \alpha\xi$ where α is either the yaw or pitch angular speed. Therefore, the force and moment drag terms due to angular displacements are

$$F_{d,angular} = -\tfrac{1}{2}\int_{x_t}^{x_n} \rho c_d \alpha\xi|\alpha\xi| \cdot 2\pi R(\xi)d\xi, \quad M_{d,angular} = -\int_{x_t}^{x_n} \tfrac{1}{2}\rho c_d \alpha\xi|\alpha\xi| \cdot \xi \cdot 2\pi R(\xi)d\xi \,.$$

Next, in order to properly evaluate the crossflow drag contribution, one must include the pressure drag of the fins (since they are bluff objects there is no skin contribution when the flow is perpendicular to the fin). This is done by replacing the

wetted area in Equation 3.28 with the projected area of the fin. Noting the surface symmetry of the RUV class from Figure 3.1, the quadratic crossflow drag force and moment coefficients arising from sway and heave velocities are therefore

$$\begin{aligned} Y_{v|v|} &= Z_{w|w|} = -\tfrac{1}{2}\rho c_{dc}\int_{x_t}^{x_n} 2\pi R(x)dx - \rho c_{df} A_{fin} \\ M_{w|w|} &= -N_{v|v|} = \tfrac{1}{2}\rho c_{dc}\int_{x_t}^{x_n} 2\pi x R(x)dx + \rho c_{df} l_{fin} A_{fin} \end{aligned} \tag{3.30}$$

where x_t is the location of the RUV tail, x_n is the location of the nose, c_{dc} is the total crossflow drag coefficient, l_{fin} is the fin moment arm, c_{df} is the fin pressure drag coefficient, and A_{fin} the frontal fin area. In like manner the quadratic crossflow drag force and moment coefficients arising from yaw and pitch angular velocities are

$$\begin{aligned} Y_{r|r|} &= -Z_{q|q|} = -\tfrac{1}{2}\rho c_{dc}\int_{x_t}^{x_n} 2\pi x|x| R(x)dx + \rho c_{df} l_{fin}^2 A_{fin} \\ M_{q|q|} &= N_{r|r|} = -\tfrac{1}{2}\rho c_{dc}\int_{x_t}^{x_n} 2\pi x^2 |x| R(x)dx - \rho c_{df} l_{fin}^3 A_{fin} \end{aligned} \tag{3.31}$$

An equation approximating the rolling drag, assuming the primary contribution results from the product of the crossflow drag of fins, $\rho c_{df} A_{fin}$, and the cube of the mean fin radius, is given by

$$K_{p|p|} = -\rho c_{df} A_{fin} \overline{r}_f^{\,3} \quad . \tag{3.32}$$

Finally, the component form of the drag forces and moments are

$$\begin{aligned} X_D &= X_{u|u|} |u| u \\ Y_D &= Y_{v|v|} |v| v + Y_{r|r|} |r| r \\ Z_D &= Z_{w|w|} |w| w + Z_{q|q|} |q| q \\ K_D &= K_{p|p|} |p| p \\ M_D &= M_{w|w|} |w| w + M_{q|q|} |q| q \\ N_D &= N_{v|v|} |v| v + N_{r|r|} |r| r \end{aligned} \tag{3.33}$$

with the coefficients evaluated as above after specifying the drag coefficients and geometry for a specific vehicle design. A more comprehensive analysis would include the effects of lift induced drag into the total drag forces and moments equations since total drag is the resultant of skin-friction, form and lift induced drag [Fox].

Body and Fin Lift Forces and Moments

At a small attack angle a body immersed in a fluid media will experience a hydrodynamic force perpendicular to the fluid motion [Fox]. The lift coefficient is defined as

$$c_L = \frac{F_L}{\frac{1}{2}\rho A_p u^2} \tag{3.34}$$

where F_L body/fin lift force, u is the fluid speed, A_p is the hull/fin planform area and ρ is the fluid density. The lift coefficient is typically a function of the angle of attack and the Reynolds number [Fox]. The angles and moments involved in the *body* lift calculations are shown in Figure 3.3. Specifically, Figure 3.3 shows the body lift attack angles (α, β), moment arm, l_{cp}, and the resulting forces (Y_{BL}, Z_{BL}) and moments (M_{BL}, N_{BL}). The schematic shows only the body lift forces generated perpendicular to the surge motion. For small attack angles, the parallel component will be negligible. Also shown are the RUV frame linear and angular velocity directions. To be precise, the body lift angle of attack is the angle between the body and the free-stream velocity vector and the fin lift attack angle is defined as the angle between the fin chord and the free-stream velocity vector. To simplify both the body and fin lift analysis, only the lift forces generated perpendicular to the surge motion are considered. For small attack angles, the parallel component will be negligible.

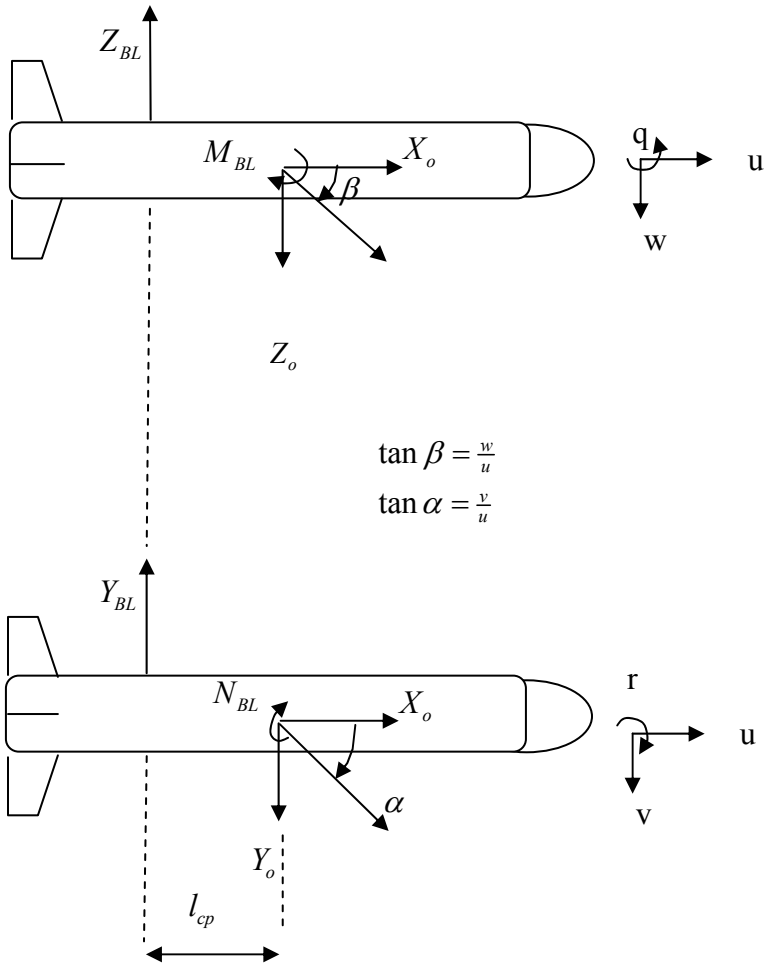

Figure 3.3. Definitions of the body lift nomenclature.

Under the slender body and small attack angle assumptions, [Triantafyllou] shows the body lift coefficients are given by

$$c_{LY_o} \simeq .171\frac{v}{u},\ c_{LZ_o} \simeq .171\frac{w}{u}. \tag{3.35}$$

Thus the respective body lift forces are given by

$$\begin{aligned} Y_{BL} &= -\tfrac{1}{2}\rho A_p(.171)uv \equiv Y_{uvl}uv \\ Z_{BL} &= -\tfrac{1}{2}\rho A_p(.171)uw \equiv Z_{uwl}uw \end{aligned}. \tag{3.36}$$

Since these forces are applied at the center of pressure, body lift moments arise. For the specified geometry the moments are

$$\begin{aligned} N_{BL} &= -Y_{BL}l_{cp} = \tfrac{1}{2}\rho A_p(.171)\cdot l_{cp}uv \equiv N_{uvl}uv \\ M_{BL} &= Z_{BL}l_{cp} = -\tfrac{1}{2}\rho A_p(.171)\cdot l_{cp}uw \equiv M_{uwl}uw \end{aligned}. \tag{3.37}$$

Here l_{cp} is the distance to the center of pressure given by [Prestero]

$$l_{cp} = 0.65l - x_n.$$

Pitch and yaw of the robot motion can be modified by adjusting the rudder and stern plane angles which are limited in their total angular travel. This is achieved because

by increasing control surface angles, the lift forces over the surfaces are increased. Since these surfaces are located a distance from the center of gravity of the vehicle, moments are induced that cause the vehicle to rotate about the corresponding axis. To calculate fin lift the effective angles and speeds need to be defined:

$$\begin{aligned} u_{fin} &= u \\ v_{fin} &= v - l_{fin} r \\ w_{fin} &= w + l_{fin} q \\ \delta_{re} &= \delta_r - \beta_{re} = \delta_r - \tfrac{1}{u}\left(v - l_{fin} r\right) \\ \delta_{se} &= \delta_s + \beta_{se} = \delta_s + \tfrac{1}{u}\left(w + l_{fin} q\right) \end{aligned} \quad . \tag{3.38}$$

Figure 3.4 shows the relation between the nomenclature and the vehicle fin forces and moments. The Hoerner empirical fin lift formula [Triantafyllou] is

$$c_r = c_s = \left(\tfrac{1}{1.8\pi} + \tfrac{1}{AR_e \pi}\right)$$

where the fin aspect ratio is given by

$$AR_e = \frac{h_{fin}^2}{A_{fin}} .$$

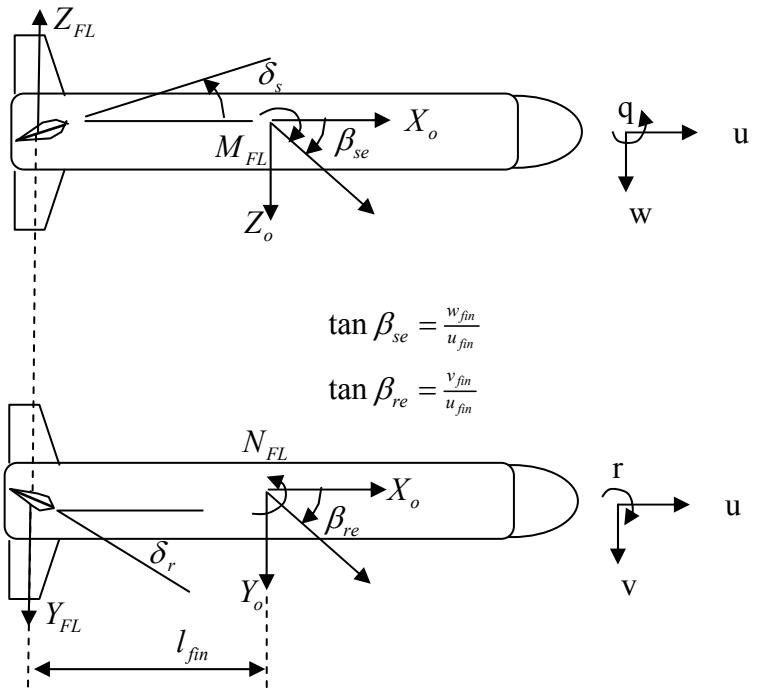

Figure 3.4. Definitions of the fin lift attack angles for the y- and z-direction fin lift forces in the vehicle reference frame. Also shown are the respective attack angles and moment arm lfin.

The y-component of the fin lift force is proportional to the product of the effective fin angle and the fin's surge speed

$$Y_{FL} = 2\left(\tfrac{1}{2}\rho A_r c_r \delta_{re} u_{fin}^2\right). \tag{3.39}$$

Substituting for δ_{re} from above gives

$$
\begin{aligned}
Y_{FL} &= 2\left(\tfrac{1}{2}\rho A_r c_r\left(\delta_r - \tfrac{1}{u}\left(v - l_{fin} r\right)\right)u^2\right) \\
&= \left(\rho A_r c_r\right)u^2\delta_r + \left(-\rho A_r c\right)uv + \left(\rho A_r c_r l_{fin}\right)ur \,. \\
&\equiv Y_{uu\delta_r}u^2\delta_r + Y_{uvf}uv + Y_{urf}ur
\end{aligned}
\tag{3.40}
$$

Similarly for the z-component yields

$$
\begin{aligned}
Z_{FL} &= 2\left(-\tfrac{1}{2}\rho A_s c_s\left(\delta_s + \tfrac{1}{u}\left(w + l_{fin} q\right)\right)u^2\right) \\
&= \left(-\rho A_s c_s\right)u^2\delta_s + \left(-\rho A_s c_s\right)uw + \left(-\rho A_s c_s l_{fin}\right)uq \\
&\equiv Z_{uu\delta_s}u^2\delta_s + Z_{uwf}uw + Z_{uqf}uq
\end{aligned}
\tag{3.41}
$$

with the force coefficients being defined in the obvious way. To obtain the moments due to the fin forces define

$$
\begin{aligned}
M_{uu\delta_s} &= M_{uwf} = -\rho A_s c_s l_{fin} \\
M_{uqf} &= -\rho A_s c_s l_{fin}^2 \\
N_{uu\delta_r} &= -N_{uvf} = -\rho A_r c_r l_{fin} \\
N_{urf} &= -\rho A_r c_r l_{fin}^2
\end{aligned}
\,.
\tag{3.42}
$$

Thus, the fin lift moments are expressed as

$$\begin{aligned} M_{FL} &= Z_{FL} l_{fin} \\ &= M_{uu\delta_s} u^2 \delta_s + M_{uwf} uw + M_{uqf} uq \\ N_{FL} &= -Y_{FL} l_{fin} \\ &= N_{uu\delta_r} u^2 \delta_r + N_{uvf} uv + N_{urf} ur \end{aligned} . \tag{3.43}$$

Finally note that the stern and rudder angles obey the inequality

$$|\delta_s| \le \Delta_s, \quad |\delta_r| \le \Delta_r$$

where Δ_s and Δ_r are typically less than 15°.

Propulsion

The model utilizes a simplified propulsion model which assumes the propeller thrust, X_{prop}, is fixed at a value such that the surge speed , u, is equal to a constant nominal value u_0. Thus $\dot{u} = 0$, and the desired thrust force can be solved for using the first of Equations (3.12) in the rigid body equations derivation section. A simplified thrust model assumes the propeller torque and thrust are linearly related

$$K_{prop} = -R \cdot X_{prop} \tag{3.44}$$

where R is the propeller radius.

Finally, the total external forces and moments constituting the right hand side of Equation 3.12 is the sum of the following contributions:

1. Added Mass, Equation 3.18: $X_A, Y_A, Z_A, K_A, M_A, N_A$
2. Hydrostatic, Equations 3.20 and 3.24: $X_{HS}, Y_{HS}, Z_{HS}, K_{HS}, M_{HS}, N_{HS}$
3. Drag, Equation 3.33: $X_D, Y_D, Z_D, K_D, M_D, N_D$
4. Body lift, Equations 3.36 and 3.37: $Y_{BL}, Z_{BL}, M_{BL}, N_{BL}$
5. Fin lift, Equations 3.40, 3.41 and 3.43: $Y_{FL}, Z_{FL}, M_{FL}, N_{FL}$
6. Propulsion, Equation 3.44: X_{prop}, K_{prop}

Full 6 Degree of Freedom RUV Equations of Motion

Therefore using the preceding analysis and summing in component form, one obtains

$$
\begin{aligned}
\sum X_{ext} &= X_{HS} + X_{u|u|}u|u| + X_{\dot{u}}\dot{u} + Z_{\dot{w}}wq + Z_{\dot{q}}q^2 - Y_{\dot{v}}vr - Y_{\dot{r}}r^2 + X_{prop} \\
\sum Y_{ext} &= Y_{HS} + Y_{v|v|}v|v| + Y_{r|r|}r|r| + Y_{\dot{v}}\dot{v} + Y_{\dot{r}}\dot{r} - Z_{\dot{w}}wp - Z_{\dot{q}}qp + Y_{ur}ur + Y_{uv}uv + Y_{uu\delta_r}u^2\delta_r \\
\sum Z_{ext} &= Z_{HS} + Z_{w|w|}w|w| + Z_{q|q|}q|q| + Z_{\dot{w}}\dot{w} + Z_{\dot{q}}\dot{q} + Y_{\dot{v}}vp + Y_{\dot{r}}rp + Z_{uq}uq + Y_{uw}uw + Y_{uu\delta_s}u^2\delta_s \\
\sum K_{ext} &= K_{HS} + K_{p|p|}p|p| + K_{\dot{p}}\dot{p} + \left(Z_{\dot{w}} - Y_{\dot{v}}\right)vw + \left(Z_{\dot{q}} + Y_{\dot{r}}\right)qv - \left(Y_{\dot{r}} + Z_{\dot{q}}\right)rw + (N_{\dot{r}} - M_{\dot{q}})qr + K_{prop} \\
\sum M_{ext} &= M_{HS} + M_{w|w|}w|w| + M_{q|q|}q|q| + Z_{\dot{q}}\dot{w} + M_{\dot{q}}\dot{q} + M_{uq}uq - Y_{\dot{r}}vp + M_{uw}uw + \left(K_{\dot{p}} - N_{\dot{r}}\right)rp + M_{uu\delta_s}u^2\delta_s \\
\sum N_{ext} &= N_{HS} + N_{v|v|}v|v| + N_{r|r|}r|r| + Y_{\dot{r}}\dot{v} + N_{\dot{r}}\dot{r} + N_{uv}uv + N_{ur}ur + Z_{\dot{q}}wp + \left(M_{\dot{q}} - K_{\dot{p}}\right)pq + N_{uu\delta_r}u^2\delta_r
\end{aligned}
\tag{3.45}
$$

where the following substitutions have been made for clarity

$$
\begin{aligned}
Y_{uv} &\equiv Y_{uvl} + Y_{uvf} \\
Y_{ur} &\equiv X_{\dot{u}} + Y_{urf} \\
Z_{uq} &\equiv -X_{\dot{u}} + Z_{uqf} \\
Z_{uw} &\equiv Z_{uwl} + Z_{uwf} \\
M_{uq} &\equiv M_{uqf} - Z_{\dot{q}} \\
M_{uw} &\equiv M_{uwf} + M_{uwl} - (Z_{\dot{w}} - X_{\dot{u}}) \\
N_{ur} &\equiv N_{urf} + Y_{\dot{r}} \\
N_{uv} &\equiv N_{uvf} + N_{uvl} + (Y_{\dot{v}} - X_{\dot{u}})
\end{aligned}
$$

Note that the terms $(Y_{\dot{v}} - X_{\dot{u}})$ and $-(Z_{\dot{w}} - X_{\dot{u}})$ represents the Munk moment coefficients [Jones, et. al.], [Triantafyllou and Hover]. Finally, the full nonlinear equations of motion become

$$
\begin{aligned}
M\dot{\nu} &= f(\eta,\nu) + g(\nu)U \\
\dot{\eta} &= J(\eta)\nu
\end{aligned}
\tag{3.46}
$$

where J and (ν, η) are defined in Equations (3.3) and (3.1), respectively, and $f \in \mathbb{R}^{6x1}, g \in \mathbb{R}^{6x3}$ are given by

$$
f(\eta,\nu) = \begin{bmatrix}
-m\left(wq - vr - x_g\left(r^2 + q^2\right) + y_g qp + z_g rp\right) + X_{HS} + X_{u|u|}u|u| + Z_{\dot{w}}wq + Z_{\dot{q}}q^2 - Y_{\dot{v}}vr - Y_{\dot{r}}r^2 \\
-m\left(ur - pw + x_g pq - y_g\left(r^2 + p^2\right) + z_g rq\right) + Y_{HS} + Y_{v|v|}v|v| + Y_{r|r|}r|r| - Z_{\dot{w}}wp - Z_{\dot{q}}qp + Y_{ur}ur + Y_{uv}uv \\
-m\left(vp - qu + x_g rp + y_g rq - z_g\left(q^2 + p^2\right)\right) + Z_{HS} + Z_{w|w|}w|w| + Z_{q|q|}q|q| + Y_{\dot{v}}vp + Y_{\dot{r}}rp + Z_{uq}uq + Z_{uw}uw \\
-I_{xy}pr + I_{xz}pq - I_{yz}\left(r^2 - q^2\right) - \left(I_z - I_y\right)rq - m\left(y_g\left(vp - uq\right) - z_g\left(ur - wp\right)\right) + K_{HS} + K_{p|p|}p|p| + \left(Z_{\dot{w}} - Y_{\dot{v}}\right)wv + \left(Z_{\dot{q}} + Y_{\dot{r}}\right)vq - \left(Y_{\dot{r}} + Z_{\dot{q}}\right)rw + \left(N_{\dot{r}} - M_{\dot{q}}\right)rq \\
I_{xy}qr - I_{yz}pq + I_{xz}\left(r^2 - p^2\right) - \left(I_x - I_z\right)pr - m\left(z_g\left(wq - vr\right) - x_g\left(vp - uq\right)\right) + M_{HS} + M_{w|w|}w|w| + M_{q|q|}q|q| + M_{uq}uq - Y_{\dot{r}}vp + M_{uw}uw + (K_{\dot{p}} - N_{\dot{r}})rp \\
I_{xz}qr + I_{yz}rp + I_{xy}\left(p^2 + q^2\right) + \left(I_x - I_y\right)pq - m\left(x_g\left(ur - wp\right) - y_g\left(wq - vr\right)\right) + N_{HS} + N_{v|v|}v|v| + N_{r|r|}r|r| + N_{uv}uv + Y_{\dot{r}}ur + Z_{\dot{q}}wp + \left(M_{\dot{q}} - K_{\dot{p}}\right)pq
\end{bmatrix}
\tag{3.47}
$$

$$g(\nu)=\begin{bmatrix} 0 & 0 & 1 \\ Y_{uu\delta_r}u^2 & 0 & 0 \\ 0 & Z_{uu\delta_s}u^2 & 0 \\ 0 & 0 & -R \\ 0 & M_{uu\delta_s}u^2 & 0 \\ N_{uu\delta_r}u^2 & 0 & 0 \end{bmatrix}, U=\begin{bmatrix} \delta_r \\ \delta_s \\ X_{prop} \end{bmatrix}. \tag{3.48}$$

Note that the simplified thrust model assumes that propeller thrust is the independent input, when in practice propeller angular speed is the actual input. The mass coefficient matrix is defined as

$$M=\begin{bmatrix} m-X_{\dot{u}} & 0 & 0 & 0 & mz_g & -my_g \\ 0 & \left(m-Y_{\dot{v}}\right) & 0 & -mz_g & 0 & \left(mx_g-Y_{\dot{r}}\right) \\ 0 & 0 & \left(m-Z_{\dot{w}}\right) & my_g & -\left(m+Z_{\dot{q}}\right) & 0 \\ 0 & -mz_g & my_g & \left(I_x-K_{\dot{p}}\right) & -I_{xy} & -I_{xz} \\ mz_g & 0 & -\left(mx_g+Z_{\dot{q}}\right) & -I_{xy} & \left(I_y-M_{\dot{q}}\right) & -I_{yz} \\ -my_g & \left(mx_g-Y_{\dot{r}}\right) & 0 & -I_{xy} & -I_{yz} & \left(I_z-N_{\dot{r}}\right) \end{bmatrix} \tag{3.49}$$

MODEL TESTING

The purpose of this section is to show through simulation that the mathematical model describing the dynamics of the RUV is realistic. It is not the intention here to establish a rigorous system identification. The model must only present a level of fidelity useful for control engineering purposes. The simulation is based on the commercially available RUV REMUS 100. In the following sections, tables for the required physical parameters for the REMUS simulation are given. The linearized state equations used in the linear control development are derived. The next sections develop the depth PID and heading PD controllers. The final section gives the results of the model verification process where simulation results are compared to actual data. Measured values for selected states were conveyed to the author directly from Woods Hole Institute by REMUS researchers. Although the exact control algorithms were not received, their approximate structure was intimated enough to reproduce with some guesswork regarding the PID structure and trajectory pre-filter [Allen, et. al., 1997].

Vehicle Parameters

In its standard configuration the REMUS (Figure 1) is 1.33 meters in length and 0.2 m in diameter. Built for long range autonomous operations, the robot hull shape is based on the Myring B hull profile in order to reduce drag and thereby enable greater mission time [Allen, et. al., 2000]. Table 3.1 lists the pertinent vehicle geometric and mass properties for the simulation. The values are taken from [Prestero]. Tables 3.2 and 3.3 list the force and moment coefficients calculated using the formulas developed in

previous sections. The y-component center of gravity requires a slightly negative value to adjust the roll angle to around 0° at a constant surge speed of 1.5 m/s. This is done in practice for single propeller vehicles so that the vehicle flies upright at its design speed.

Table 3.1. REMUS RUV Simulation Parameters

Parameter	Value	Units	Description
ρ	1010	g/m^3	Fluid density
g	9.81	m/s^2	Gravitational acceleration
x_n	0.6	m	Distance form origin to vehicle nose
x_t	-0.73	m	Distance form origin to vehicle tail
x_{fin1}	-0.7	m	Distance form origin to fin trailing edge
x_{fin2}	-0.6	m	Distance form origin to fin leading edge
d	.191	m	Max body diameter
l	1.33	m	Vehicle body length
S_w	.7981	m^2	Approximate wetted surface area (πdl)
A_p	.2540	m^2	Approximate hull planform area (ld)
A_f	.0287	m^2	Approximate frontal area ($\pi d^2/4$)
c_{ds}	0.004	-	Skin drag coefficient
c_{dF}	0.3	-	Axial drag coefficient referenced to A_f
c_{da}	.0166	-	Total axial drag coefficient ref. to S_w
c_{dc}	.0459	-	Total crossflow coefficient ref. to S_w
l_{cp}	0. 2645	m	Distance to center of pressure
$\overline{r}_f$	0.131	m	Distance from vehicle axial center to fin center
R_{fin}	0.131	m	Distance from vehicle axial center to fin tip
w_{fin}	0.0890	m	Width of fin base
	0.0590	m	Width of fin tip
t_{fin}	1010	g/m^3	Fluid density
h_{fin}	0.0960	m	Fin height
A_{fin}	0.0071	m^2	Rudder/stern fin area
A_r, A_s	0.0071	m^2	Rudder/stern fin planform area
c_{df}	1.558	-	Fin form drag coefficient
c_r, c_s	2.3685	-	Rudder/stern fin lift coefficient
l_{fin}	0.8190	m	Fin moment arm
W	3.00×10^2	N	Weight
B	3.06×10^2	N	Buoyancy
x_b	0	m	x-component center of buoyancy
y_b	0	m	y-component " " "
z_b	0	m	z-component " " "
x_g	0	m	x-component center of gravity
y_g	-0.8×10^{-2}	m	y-component " " "
z_g	1.96×10^{-2}	m	z-component " " "
I_{xx}	1.77×10^{-1}	$kg\ m^2$	Moment of inertia about x axis
I_{yy}	3.45	$kg\ m^2$	Moment of inertia about y axis
I_{zz}	3.45	$kg\ m^2$	Moment of inertia about z axis
R	5.87E-2	m	Propeller radius
u_0	1.5	m/s	Design speed

Table 3.2. REMUS Moment Coefficients

Parameter	Value	Units	Description
$K_{p\|p\|}$	-5.03	kg m^2/rad^2	Rolling drag
$K_{\dot{p}}$	-0.095	kg m^2/rad^2	Added mass
$M_{w\|w\|}$	7.95	kg	Cross flow drag
$M_{q\|q\|}$	-24.13	kg m^2/rad^2	Cross flow drag
M_{uw}	21.89	kg	Body and fin lift + Munk Moment
$M_{\dot{w}}$	5.16	kg m	Added mass
$M_{\dot{q}}$	-7.57	kg m^2/rad	Added mass
M_{uq}	-16.56	kg m/rad	Added mass cross term + fin lift
$M_{uu\delta_s}$	-13.92	kg/rad	Fin lift moment
$N_{v\|v\|}$	-7.95	kg	Cross flow drag
$N_{r\|r\|}$	-24.13	kg m^2/rad	Cross flow drag
N_{uv}	-21.89	kg	Body and fin lift + Munk Moment
$N_{\dot{v}}$	-5.16	kg m	Added mass
$N_{\dot{r}}$	-7.57	kg m^2/rad	Added mass
$N_{uu\delta_r}$	-13.92	kg rad	Fin lift moment

Table 3.3. REMUS Force Coefficients

Parameter	Value	Units	Description
$X_{u\|u\|}$	-6.68	kg/m	Axial drag
$X_{\dot{u}}$	-0.513	kg	Added mass
$Y_{v\|v\|}$	-196.26	kg/m	Cross-flow drag
$Y_{r\|r\|}$	8.30	kg m/rad^2	Cross-flow drag
Y_{uv}	-38.93	kg/m	Body and fin lift
$Y_{\dot{v}}$	-42.13	kg	Added mass
$Y_{\dot{r}}$	-5.16	kg m/rad	Added mass
Y_{ur}	13.41	kg/rad	Added mass cross term + fin lift
$Y_{uu\delta_r}$	16.99	kg/m/rad	Fin lift force
$Z_{w\|w\|}$	-196.26	kg/m	Cross flow drag
$Z_{q\|q\|}$	-8.30	kg m/rad^2	Cross flow drag
Z_{uw}	-38.93	kg/m	Body and fin lift
$Z_{\dot{w}}$	-42.13	kg	Added mass
$Z_{\dot{q}}$	5.16	kg m/rad	Added mass
Z_{uq}	-13.41	kg/rad	Added mass cross term + fin lift
$Z_{uu\delta_s}$	-16.99	kg/m/rad	Fin lift force

Open Loop Heading Instability

This section discusses the inherent heading open loop instability of the present class of RUV. Due to the Munk moment, $(Y_{\dot{v}} - X_{\dot{u}})$, in the equation for the yaw dynamics, the heading subsystem is unstable [Triantafyllou]. The instability is mitigated by vortex shedding at the vehicle stern as well as fin and body lift/drag [Triantafyllou]. There is a tradeoff between stability and maneuverability. If the vehicle is overly stable, the vehicle will require large fins to turn, increasing drag. If the vehicle is too unstable, however, excessive control action will be required, increasing power consumption. The optimization of this design tradeoff is beyond the scope of this research.

Figures 3.5 – 3.6 shows the horizontal plane motion of the RUV for a 20 second interval with the vehicle initially placed at the origin. The rudder control fins are fixed at -0.5°. The RUV's heading instability causes the vehicle to turn in a 6 meter diameter circular path in the opposite direction of the fin-induced moment.

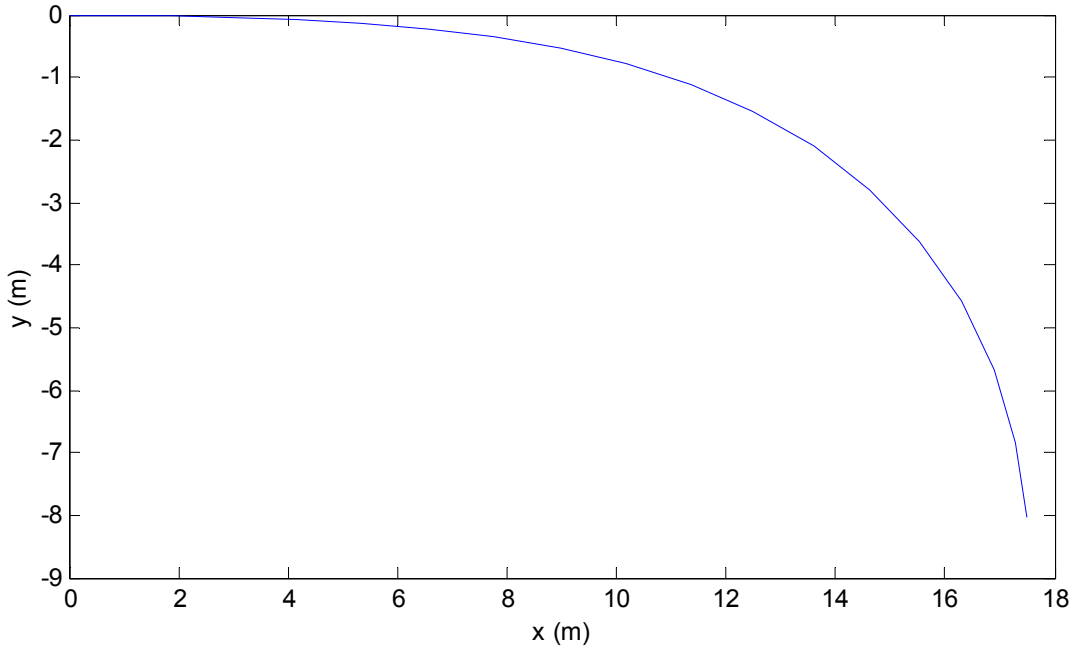

Figure 3.5. Inherent open-loop heading instability of the RUV. The graph shows the horizontal plane motion in inertial coordinates of the RUV for a 20 second period. The rudder fins are fixed at -0.5° which, in the absence of the Munk moment, would drive the vehicle in the +y-direction. However, due to the destabilizing heading Munk moment, the RUV actually turns in the direction opposite of its fin command.

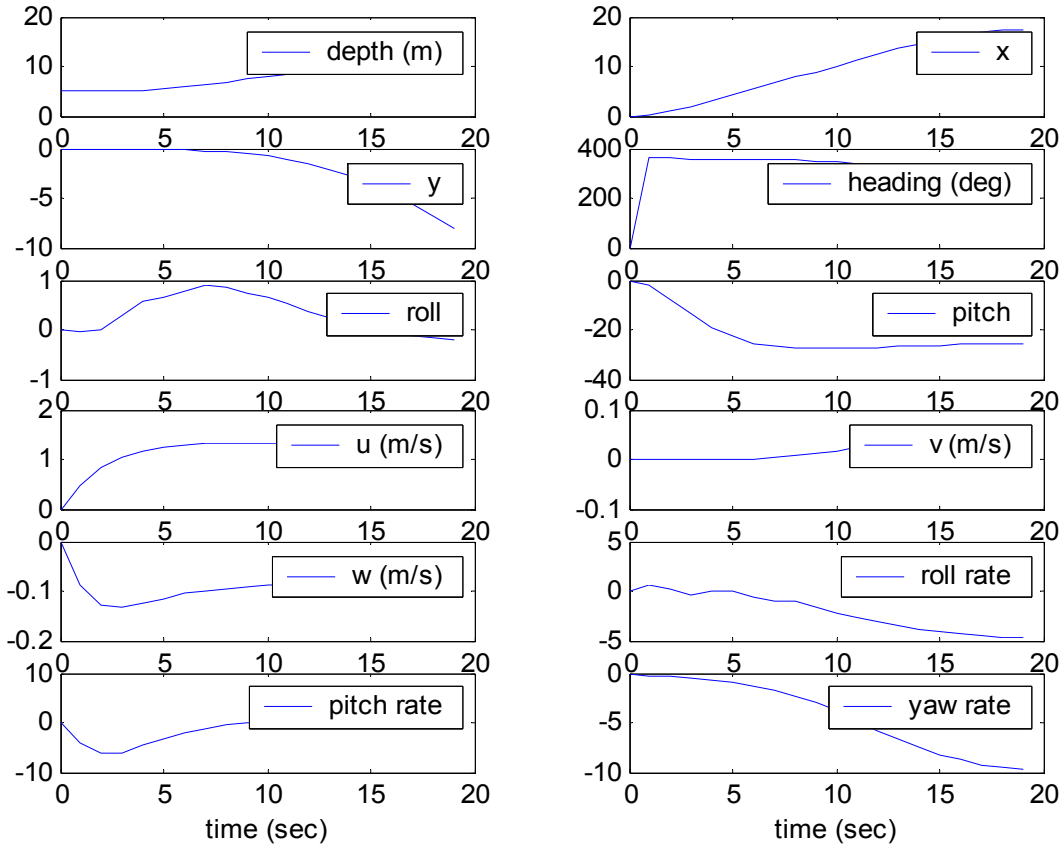

Figure 3.6. RUV states during open-loop heading maneuver. The plots show the RUV states during the open loop maneuver demonstrating the inherent heading instability of the RUV class.

Linear control

This subsection develops the RUV linearized state equations used in the subsequent derivation and application of the basic linear, decoupled PID heading and depth controller. The controller is used in this section for model verification purposes and in later sections for comparison with the nonlinear control approach. The technique reported in [Allen 1997] assumes the surge, heading and depth states can be decoupled and then uses separate controllers for depth and heading. Figure 3.7 illustrates the basic control structure in block diagram form which will be obtained in the following sections.

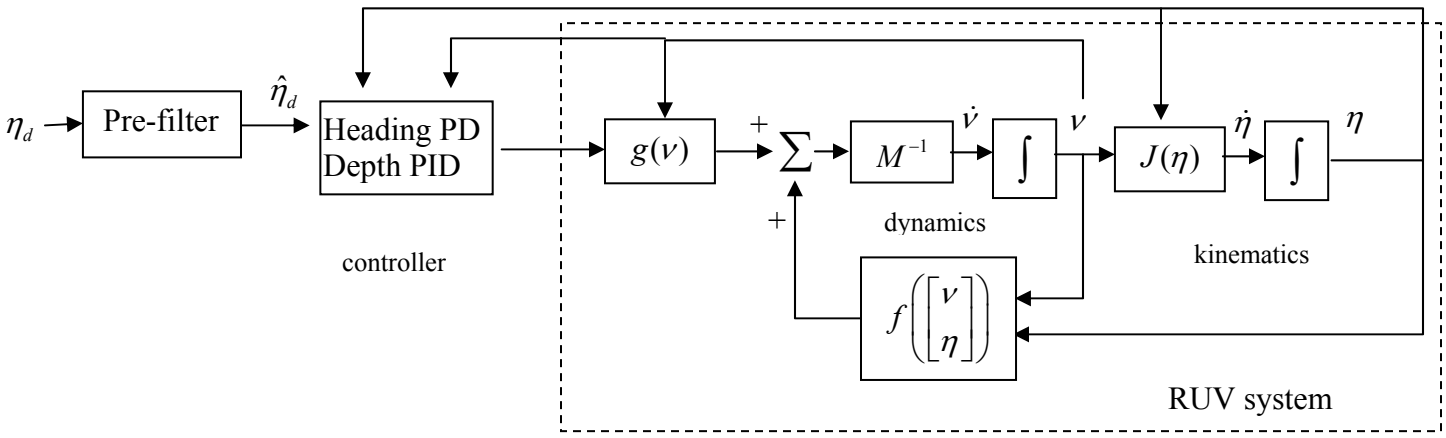

Figure 3.7. Block diagram of the closed loop nonlinear system used for the model verification effort (Equation 3.46). The controller block is comprised of decoupled heading and depth controllers. The surge speed is set in open loop mode by fixing the thruster rpm.

Deriving a linear state feedback control law first requires finding the linearization of Equation 3.46 about a specified operating trajectory. For the kinetic expression of Equation 3.46 there is

$$M\delta\dot{\nu} = \begin{bmatrix} \left.\frac{\partial f}{\partial \nu}\right|_{\nu=\nu_0} & \left.\frac{\partial f}{\partial \eta}\right|_{\eta=\eta_0} \end{bmatrix} \begin{bmatrix} \delta\nu \\ \delta\eta \end{bmatrix} + g(\nu_0)U \tag{3.50}$$

The linearized perturbation of the kinematic expression of Equation 3.46 is given by

$$\delta\dot{\eta} = \begin{bmatrix} J(\eta_0) & \frac{\partial}{\partial \eta}\left[J(\eta)\nu\right]\Big|_{(\eta_0,\nu_0)} \end{bmatrix} \begin{bmatrix} \delta\nu \\ \delta\eta \end{bmatrix} \tag{3.51}$$

Denoting the Euler velocity transformation matrix, $J(\eta_2)$, as a column of its row vectors

$$J(\eta_2) = \begin{bmatrix} J_1 \\ J_2 \\ J_3 \\ J_4 \\ J_5 \\ J_6 \end{bmatrix} \tag{3.52}$$

the *i-j*th element of the second matrix term inside the brackets of Equation 3.51 above becomes

$$\left[\frac{\partial}{\partial \eta}\left(J(\eta)\cdot\nu\right)\right]_{i,j} = \frac{\partial}{\partial \eta_j}\left(J_i\cdot\nu\right) \tag{3.53}$$

Define the matrix composed of these elements as $J^*(\eta,\nu)$. Thus, one obtains

$$\delta\dot{\eta} = \begin{bmatrix} J(\eta_0) & J^*(\eta_0,\nu_0)\end{bmatrix}\begin{bmatrix}\delta\nu \\ \delta\eta\end{bmatrix} \tag{3.54}$$

and the linearized equations of motion are given by

$$\begin{aligned}\begin{bmatrix} M\delta\dot{\nu} \\ \delta\dot{\eta}\end{bmatrix} &= \begin{bmatrix} \left.\frac{\partial f}{\partial \nu}\right|_{\nu=\nu_0} & \left.\frac{\partial f}{\partial \eta}\right|_{\eta=\eta_0} \\ J(\eta_0) & J^*(\eta_0,\nu_0)\end{bmatrix}\begin{bmatrix}\delta\nu \\ \delta\eta\end{bmatrix} + \begin{bmatrix} g(\nu_0) \\ 0_{6x3}\end{bmatrix}U \\ &\equiv A\begin{bmatrix}\delta\nu \\ \delta\eta\end{bmatrix} + BU\end{aligned} \tag{3.55}$$

where $U = \begin{bmatrix}\delta_r & \delta_s & X_{prop}\end{bmatrix}^T$. Performing the indicated operations yields the following matrices.

$$B = \begin{bmatrix} 0 & 0 & 1 \\ Y_{uu\delta_r} u_0^2 & 0 & 0 \\ 0 & Z_{uu\delta_s} u_0^2 & 0 \\ 0 & 0 & -R \\ 0 & M_{uu\delta_s} u_0^2 & 0 \\ N_{uu\delta_r} u_0^2 & 0 & 0 \\ \hline 0_{6x3} \end{bmatrix}, \quad A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \tag{3.56}$$

where the A-matrix components are

$$A_{11} = \begin{bmatrix} X_{u|u|} & (m+Y_{\dot v})r & (-m+Z_{\dot w})q & -m(y_g q + z_g r) & -m(w+y_g p) & m(v-z_g p) \\ -mr + Y_{ur} r + Y_{uv} v & Y_{v|v|} + Y_{uv} u & mp - Z_{\dot w} p & m(w - x_g q) - Z_{\dot w} w - Z_{\dot q} q & -m(x_g p + z_g r) - Z_{\dot q} p & -m(u + z_g q) + Y_{ur} u \\ mq + Z_{uq} q + Z_{uw} w & -mp + Y_{\dot v} p & Z_{w|w|} + Z_{uw} u & -m x_g r + Y_{\dot v} v + Y_{\dot r} r & m(u - y_g r) + Z_{uq} u & -m(x_g p + y_g q) + Y_{\dot r} p \\ m(y_g q + z_g r) & \begin{matrix} -m y_g p \\ +(Z_{\dot w} - Y_{\dot v}) w \\ +(Z_{\dot q} + N_{\dot v}) q \end{matrix} & \begin{matrix} -m z_g p \\ +(Z_{\dot w} - Y_{\dot v}) v \\ -(Y_{\dot r} + M_{\dot w}) r \end{matrix} & \begin{matrix} -I_{xy} r + I_{xz} q \\ -m(y_g v + z_g w) + K_{p|p|} \end{matrix} & \begin{matrix} I_{xz} p - (I_z - I_y) r \\ +m y_g u + (Z_{\dot q} + N_{\dot v}) v \\ +(N_{\dot r} - M_{\dot q}) r \end{matrix} & \begin{matrix} -I_{xy} p - (I_z - I_y) q \\ +m z_g u - (Y_{\dot r} + M_{\dot w}) w \\ +(N_{\dot r} - M_{\dot q}) q \end{matrix} \\ -m x_g q + M_{uq} q + M_{uw} w & m(z_g r + x_g p) - N_{\dot v} p & -m z_g q + M_{uw} u & \begin{matrix} -I_{yz} q - (I_x - I_z) r \\ +(m x_g - N_{\dot v}) v \\ +(K_{\dot p} - N_{\dot r}) r \end{matrix} & \begin{matrix} I_{xy} r - I_{yz} p \\ -m(z_g w + x_g u) \\ +M_{uq} u + M_{q|q|} \end{matrix} & \begin{matrix} I_{xy} q - (I_x - I_z) p \\ +m z_g v \\ +(K_{\dot p} - N_{\dot r}) p \end{matrix} \\ -m x_g r + N_{uv} v + Y_{\dot r} r & -m y_g r + N_{uv} u & \begin{matrix} m(x_g p + y_g q) \\ +M_{\dot w} p \end{matrix} & \begin{matrix} I_{yz} r + (I_x - I_y) q \\ +m x_g w + M_{\dot w} w \\ +(M_{\dot q} - K_{\dot p}) q \end{matrix} & \begin{matrix} I_{xz} r + (I_x - I_y) p \\ +m y_g w + (M_{\dot q} - K_{\dot p}) p \end{matrix} & \begin{matrix} I_{xz} q + I_{yz} p \\ -m(x_g u + y_g v) \\ +N_{r|r|} + Y_{\dot r} u \end{matrix} \end{bmatrix} \tag{3.57}$$

$$A_{12} = \begin{bmatrix} 0 & 0 & 0 & 0 & -(W-B)\cos\theta & 0 \\ 0 & 0 & 0 & -(W-B)\cos\theta\cos\phi & -(W-B)\sin\theta\sin\phi & 0 \\ 0 & 0 & 0 & -(W-B)\cos\theta\sin\phi & -(W-B)\sin\theta\cos\phi & 0 \\ 0 & 0 & 0 & -(y_g W - y_b B)\cos\phi - (z_g W - z_b B)\cos\theta\cos\phi & -(y_g W - y_b B)\sin\theta\cos\phi + (z_g W - z_b B)\sin\theta\sin\phi & 0 \\ 0 & 0 & 0 & (x_g W - x_b B)\cos\theta\sin\phi & -(z_g W - z_b B)\cos\theta + (x_g W - x_b B)\sin\theta\cos\phi & 0 \\ 0 & 0 & 0 & (x_g W - x_b B)\cos\theta\cos\phi & -(x_g W - x_b B)\sin\theta\sin\phi + (y_g W - y_b B)\cos\theta & 0 \end{bmatrix} \tag{3.58}$$

$$A_{21} = \begin{bmatrix} \cos\psi\cos\theta & -\sin\psi\cos\phi + \cos\psi\sin\theta\sin\phi & \sin\psi\sin\phi + \cos\psi\cos\phi\sin\theta & 0 & 0 & 0 \\ \sin\psi\cos\theta & \cos\psi\cos\phi + \sin\phi\sin\theta\sin\psi & -\cos\psi\sin\phi + \sin\theta\sin\psi\cos\phi & 0 & 0 & 0 \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & 0 & 0 & 0 & \cos\phi & -\sin\phi \\ 0 & 0 & 0 & 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta} \end{bmatrix}$$

(3.59)

$$A_{22} = \begin{bmatrix} 0 & 0 & 0 & v(s\psi s\phi + c\psi s\theta c\phi) + w(s\psi c\phi - c\psi s\phi s\theta) & -uc\psi s\theta + vc\psi c\theta s\phi + wc\psi c\phi c\theta & -us\psi c\theta - v(c\psi c\phi + s\psi s\theta s\phi) + w(c\psi s\phi - s\psi c\phi s\theta) \\ 0 & 0 & 0 & -v(c\psi s\phi - s\psi s\theta c\phi) - w(c\psi c\phi + s\psi s\phi s\theta) & -us\psi s\theta + vs\psi c\theta s\phi + ws\psi c\phi c\theta & uc\psi c\theta + v(-s\psi c\phi + c\psi s\theta s\phi) + w(s\psi s\phi + c\psi c\phi s\theta) \\ 0 & 0 & 0 & vc\theta c\phi - wc\theta s\phi & -uc\theta - vs\theta s\phi - ws\theta c\phi & 0 \\ 0 & 0 & 0 & qc\phi t\theta - rs\phi t\theta & (qs\phi + rc\phi)\sec^2\theta & 0 \\ 0 & 0 & 0 & -qs\phi - rc\phi & 0 & 0 \\ 0 & 0 & 0 & q\frac{c\phi}{c\theta} - r\frac{s\phi}{c\theta} & (qs\phi + rc\phi)\frac{\mathrm{t}\theta}{c\theta} & 0 \end{bmatrix}$$

(3.60)

where *sin*, *cos*, etc. have been symbolized by *s*, *c*, etc. in Equation 3.60. The foregoing state linearization assumes:

1. The damping matrix is diagonal: $D(v) = -diag\left\{X_{u|u|}, Y_{v|v|}, Z_{w|w|}, K_{p|p|}, M_{q|q|}, N_{r|r|}\right\}$
2. Terms of order 2 and higher in one variable can be neglected, e.g., u^2, v^2, etc.

Depth Autopilot Design

[Allen, et. al., 1997] reports the use of an inner loop pitch PID and outer loop depth proportional controller. Assume the depth-related states

$$x_{depth} = \begin{bmatrix} q & \theta & z \end{bmatrix}^T \tag{3.61}$$

can be decoupled from the system. From Equations (3.54)-(3.58) with

$$\begin{bmatrix} \nu_0 \\ \eta_0 \end{bmatrix} = \begin{bmatrix} u_0 & 0 & 0 & 0 & 0 & 0 & - & - & - & 0 & 0 & - \end{bmatrix}^T \tag{3.62}$$

where the dashes indicate variables which do not appear in the linearized subsystem state equations and therefore have no specified operation point. Those variables are the inertial coordinates *x, y, z* and ψ. Using Equations 3.57-3.60, the linearized equations about the constant surge speed, $u_0 = 1.5$ m/s, operating condition are:

$$\begin{bmatrix} \dot{q} \\ \dot{\theta} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} \dfrac{\left(-mx_g + M_{uq}\right)u_0 + M_{q|q|}}{I_y - M_{\dot{q}}} & \dfrac{\left(z_g W - z_b B\right)}{I_y - M_{\dot{q}}} & 0 \\ 1 & 0 & 0 \\ 0 & -u_0 & 0 \end{bmatrix} \begin{bmatrix} q \\ \theta \\ z \end{bmatrix} + \begin{bmatrix} \dfrac{M_{uu\delta_s} u_0^2}{I_y - M_{\dot{q}}} \\ 0 \\ 0 \end{bmatrix} \delta_s$$
$$= \begin{bmatrix} -5.19 & -0.53 & 0 \\ 1 & 0 & 0 \\ 0 & -1.5 & 0 \end{bmatrix} \begin{bmatrix} q \\ \theta \\ z \end{bmatrix} + \begin{bmatrix} -5.05 \\ 0 \\ 0 \end{bmatrix} \delta_s . \tag{3.63}$$

The structure of the depth PID controller shown in Figure 3.8 utilizes an inner pitch loop PID and outer depth loop proportional controller.

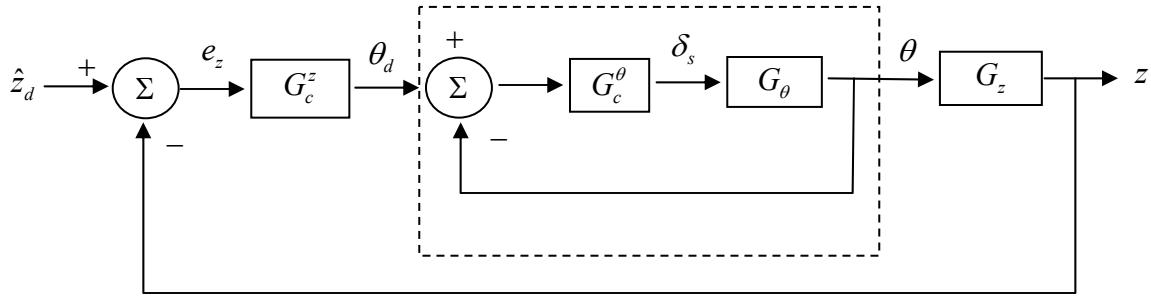

Figure 3.8 Depth controller utilizes an inner loop pitch PID and an outer loop depth proportional controller. The response of the pitch loop (dashed box) is required to be sufficiently higher than the outer loop response in order to assume $\theta_d = \theta$ in the control law derivation.

In the derivations to follow assume full subsystem measurability. The pitch and depth subsystem transfer functions are given by

$$G_\theta = C_\theta^T \left(sI - A_d \right) B_d$$

and

$$G_z = C_z^T \left(sI - A_d \right) B_d .$$

respectively. The matrices are defined from Equation 3.63:

$$\begin{bmatrix} \dot{q} \\ \dot{\theta} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} \dfrac{\left(-mx_g + M_{uq}\right)u_0 + M_{q|q|}}{I_y - M_{\dot{q}}} & \dfrac{\left(z_g W - z_b B\right)}{I_y - M_{\dot{q}}} & 0 \\ 1 & 0 & 0 \\ 0 & -u_0 & 0 \end{bmatrix} \begin{bmatrix} q \\ \theta \\ z \end{bmatrix} + \begin{bmatrix} \dfrac{M_{uu\delta_s} u_0^2}{I_y - M_{\dot{q}}} \\ 0 \\ 0 \end{bmatrix} \delta_s$$

$$\equiv A_d \begin{bmatrix} q \\ \theta \\ z \end{bmatrix} + B_d \delta_s .$$

where the system matrix components are

$$a_{11,d} = \frac{\left(-mx_g + M_{uq}\right)u_0 + M_{q|q|}}{I_y - M_{\dot{q}}}$$

$$a_{12,d} = \frac{-z_g W - z_b B}{I_y - M_{\dot{q}}}$$

$$b_{1,d} = \frac{M_{uu\delta_s} u_0^2}{I_y - M_{\dot{q}}}$$

Recalling that $x_{depth} = \begin{bmatrix} q & \theta & z \end{bmatrix}^T$, the measurement coefficient vectors are

$$C_\theta = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T , C_z = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T .$$

The system transfer functions are

$$G_\theta = \frac{b_{1,d}}{s^2 - a_{11,d}s - a_{12,d}}$$

and

$$G_z = \frac{u_0}{s}\frac{b_{1,d}}{s^2 - a_{11,d}s - a_{12,d}} = \frac{u_0}{s}G_\theta .$$

If the inner loop pitch and depth controller transfer functions are

$$G_c^\theta = K_p^\theta + K_i^\theta / s + K_d^\theta s,\ G_c^z = K_p^z$$

respectively, then the closed loop transfer functions are

$$T_\theta = \frac{\left(K_d^\theta s^2 + K_p^\theta s + K_i^\theta\right)b_{1,d}}{s^3 + \left(b_{1,h}K_d^\theta - a_{11,d}\right)s^2 + \left(b_{1,d}K_p^\theta - a_{12,d}\right)s + b_{1,d}K_i^\theta} \tag{3.64}$$

and

$$T_z = \frac{u_0 K_p^z}{s + u_0 K_p^z},$$

respectively, where it is assumed that the pitch response is sufficiently faster than the depth response so that $\theta = \theta_d$ (see Figure 3.8, dashed box). Equating the ITAE optimal coefficients to the denominator coefficients in Equation 3.64 above yields the pitch loop gains

$$K_p^\theta = \frac{a_{12,d} + 215}{b_{1,d}}, \quad K_i^\theta = \frac{1000}{b_{1,d}}, \quad K_d^\theta = \frac{17.5 + a_{11,d}}{b_{1,d}}.$$

If one assumes a 6 second time constant for the depth outer loop, the proportional depth gain is given by

$$K_p^z = \frac{1}{6u_0}.$$

From this analysis the outer loop controller gain is $K_p^z = 0.0833$ while the inner loop pitch gains are $K_p^\theta = -1.064e3$, $K_i^\theta = -2.47e4$ and $K_d^\theta = -16.29$. In order to eliminate the closed loop transfer function zeros and improve overshoot, the depth reference state is filtered according to the dynamics

$$\ddot{\hat{z}}_d + 2\zeta\omega\dot{\hat{z}}_d + \omega^2\hat{z}_d = \omega^2 z_d \tag{3.65}$$

with $\zeta = 0.8$ and $\omega = 0.25$. While the prefilter parameters could be calculated analytically [Dorf and Bishop], a more successful approach is to tune the parameters experimentally according to the measured response. This is because the derived transfer function assumes the depth states are decoupled. However unmodelled dynamics will effect the exact location of the zeros in the neighborhood of the operating surge speed, u_0.

Heading Autopilot Design

[Allen, et. al., 1997] reports the use of a heading PID, however it was necessary to utilize a simpler PD as the integral term contributed to plant instability. Assume the heading related states can be can be decoupled from the system. The heading substate is

$$x_{heading} = \begin{bmatrix} r & \psi \end{bmatrix}^T . \tag{3.66}$$

Using the above linearized equations and nominal state as before (Equation 3.62) there results

$$\begin{aligned} \begin{bmatrix} \dot{r} \\ \dot{\psi} \end{bmatrix} &= \begin{bmatrix} \dfrac{-mx_g u_0 + N_{r|r|} + Y_{\dot{r}} u_0}{I_z - N_{\dot{r}}} & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} r \\ \psi \end{bmatrix} + \begin{bmatrix} \dfrac{N_{uu\delta_r} u_0^2}{I_z - N_{\dot{r}}} \\ 0 \end{bmatrix} \delta_r \\ &= \begin{bmatrix} -3.13 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} r \\ \psi \end{bmatrix} + \begin{bmatrix} -5.05 \\ 0 \end{bmatrix} \delta_r \end{aligned} \tag{3.67}$$

Again assume full state measurability. The structure of the heading PD controller is shown in Figure 3.9.

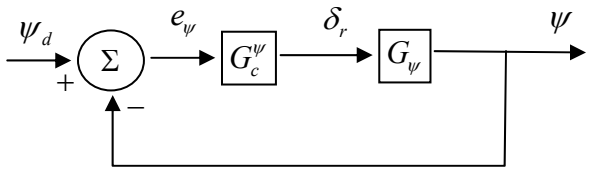

Figure 3.9 RUV heading controller utilizing a PD control law.

Following the above approach to finding the transfer function and recalling that $x_{heading} = \begin{bmatrix} r & \psi \end{bmatrix}^T$, one obtains

$$G_\psi = C_\psi^T \left(sI - A_h \right) B_h$$

where $C_\psi = \begin{bmatrix} 0 & 1 \end{bmatrix}^T$. The required system matrices can be found from Equation 3.64.

$$\begin{bmatrix} \dot{r} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} \dfrac{-mx_g u_0 + N_{r|r|} + Y_{\dot{r}} u_0}{I_z - N_{\dot{r}}} & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} r \\ \psi \end{bmatrix} + \begin{bmatrix} \dfrac{N_{uu\delta_r} u_0^2}{I_z - N_{\dot{r}}} \\ 0 \end{bmatrix} \delta_r$$
$$= \begin{bmatrix} a_{11,h} & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} r \\ \psi \end{bmatrix} + \begin{bmatrix} b_{1,h} \\ 0 \end{bmatrix} \delta_r$$
$$\equiv A_h \begin{bmatrix} r \\ \psi \end{bmatrix} + B_h \delta_r .$$

Therefore the plant transfer function is

$$G_\psi = \frac{b_{1,h}}{s\left(s - a_{11,h}\right)} .$$

The PD controller transfer functions is

$$G_c^\psi = K_p^\psi + K_d^\psi s .$$

Thus the closed loop transfer function is

$$T_\psi = \frac{\left(K_d^\psi s + K_p^\psi\right) b_{1,h}}{s^2 + \left(b_{1,h} K_d^\psi - a_{11,h}\right) s + b_{1,h} K_p^\psi} .$$

Equating the denominator coefficients to the standard 2nd order system transfer function one obtains

$$K_d^{\psi} = \frac{2\zeta + a_{11,h}}{b_{1,h}}, \quad K_p^{\psi} = \frac{\omega_n^2}{b_{1,h}}.$$

where ζ and ω_n are the damping coefficient and design frequency.

The foregoing analysis yields the controller gains $K_{p,heading} = -10.47$ and $K_{d,heading} = -2.55$ for ζ = 1.1 and ω_n = 7.27 . As a means of tuning the model to the measured data, the depth reference state is filtered according to the dynamics

$$\ddot{\hat{\psi}}_d + 2\zeta\omega\dot{\hat{\psi}}_d + \omega^2\hat{\psi}_d = \omega^2\psi_d$$

with ζ = 1.1 and ω = 1.1. Here ψ_d is the desired heading and $\hat{\psi}_d$ is the filtered desired heading. The filter helps the RUV closed-loop response eliminate unwanted dynamics associated with the desired trajectory transfer function.

Digital Implementation

The continuous time-domain form of the general PID control law is given by

$$u(t) = K_p e(t) + K_i \int_{-\infty}^{t} e(\tau)d\tau + K_d \dot{e}(t)$$

where *e(t)* is the error. In the discrete domain the individual terms are approximated as follows. The proportional term is simply

$$u_p(k) = K_p e(k).$$

The integral and derivative terms are respectively approximated by

$$u_i(k) = K_i T_s \sum_{i=0}^{k} e(i) = K_i T_s e(k) - u_i(k-1)$$

and

$$u_d(k) = \frac{K_d}{T_s}\big(e(k) - e(k-1)\big).$$

Closed Loop Model Test Results

This subsection shows the results of the linearized decoupled control model verification results for the REMUS RUV model presented above The simulation is based on the true (nonlinear) dynamics. In this section it is shown that the model is in good agreement with the measure states. Chapter 4 will examine its performance in comparison to the regular form sliding mode controller. The robot state response to the

depth and heading input commands are shown in Figures 3.10-3.12 below. The measured responses were available for selected state variables and are shown in green. The data are taken under nominal flight conditions during a sonar system evaluation maneuver. Aside from initial measurement noise due to launch perturbations and surface wave action, there appears to be acceptable agreement between simulation and measurement. Furthermore, the nonmeasured states agree with physical reality of the vehicle motion, i.e., average sway and yaw velocities are practically zero and pitch and roll rates are within acceptable operating limits of the vehicle. For example, the RUV is designed to shut itself down if the pitch angle ever increases beyond a value of ±30° from the horizontal. The present analysis establishes a reasonable similitude between the simulated and actual *closed loop* behavior of a similar RUV.

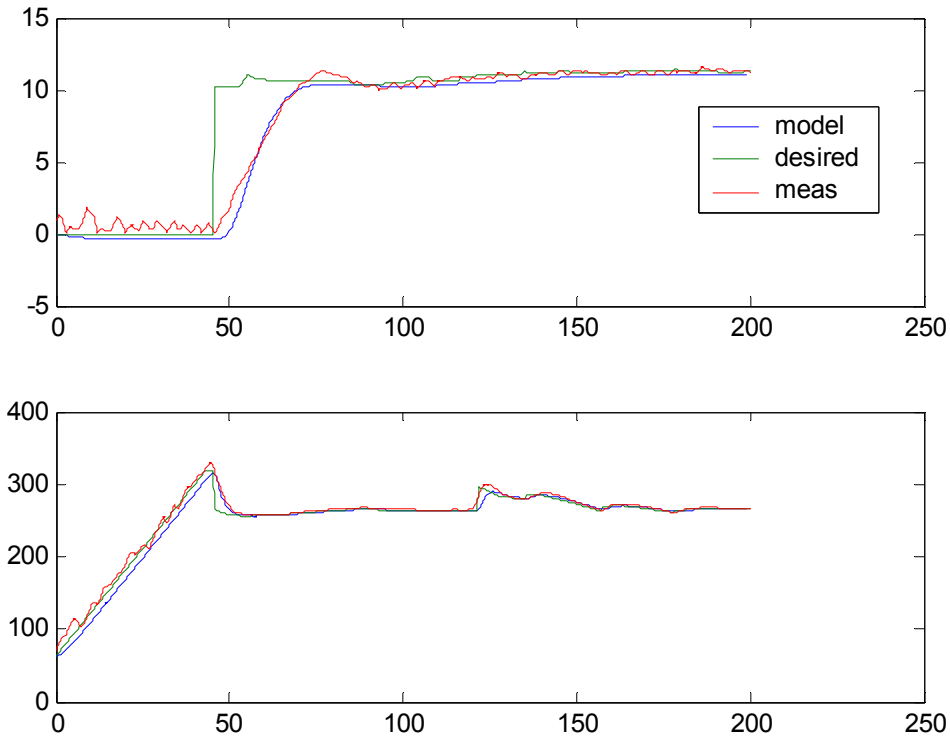

Figure 3.10. Simulated and measured depth (top) and heading (bottom) response of the REMUS RUV. The desired response is shown as the green data series. The control loop rate is nominally 5 Hz.

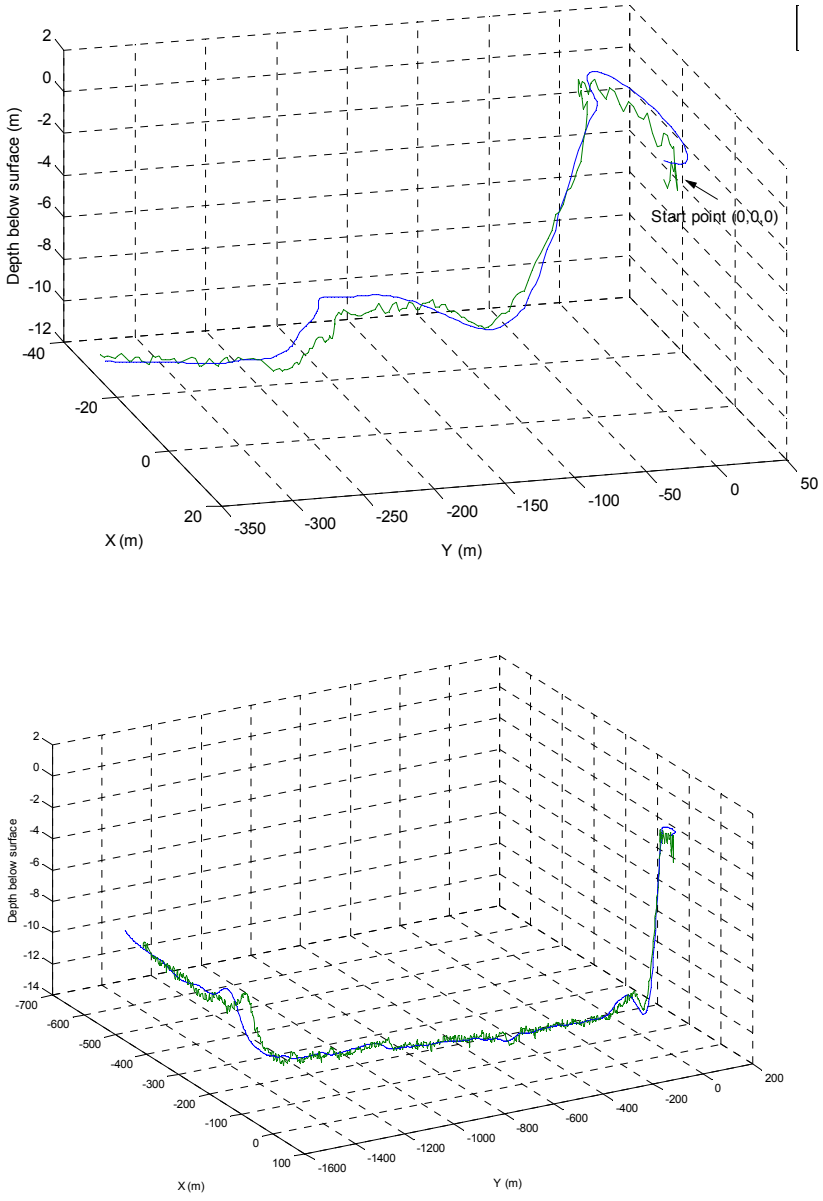

Figure 3.11. REMUS Earth-frame position for the linear control model verification. Simulated (blue) and measured (green) position of the REMUS RUV in meters over a period of approximately 3 minutes (top) and fifteen minutes (bottom).

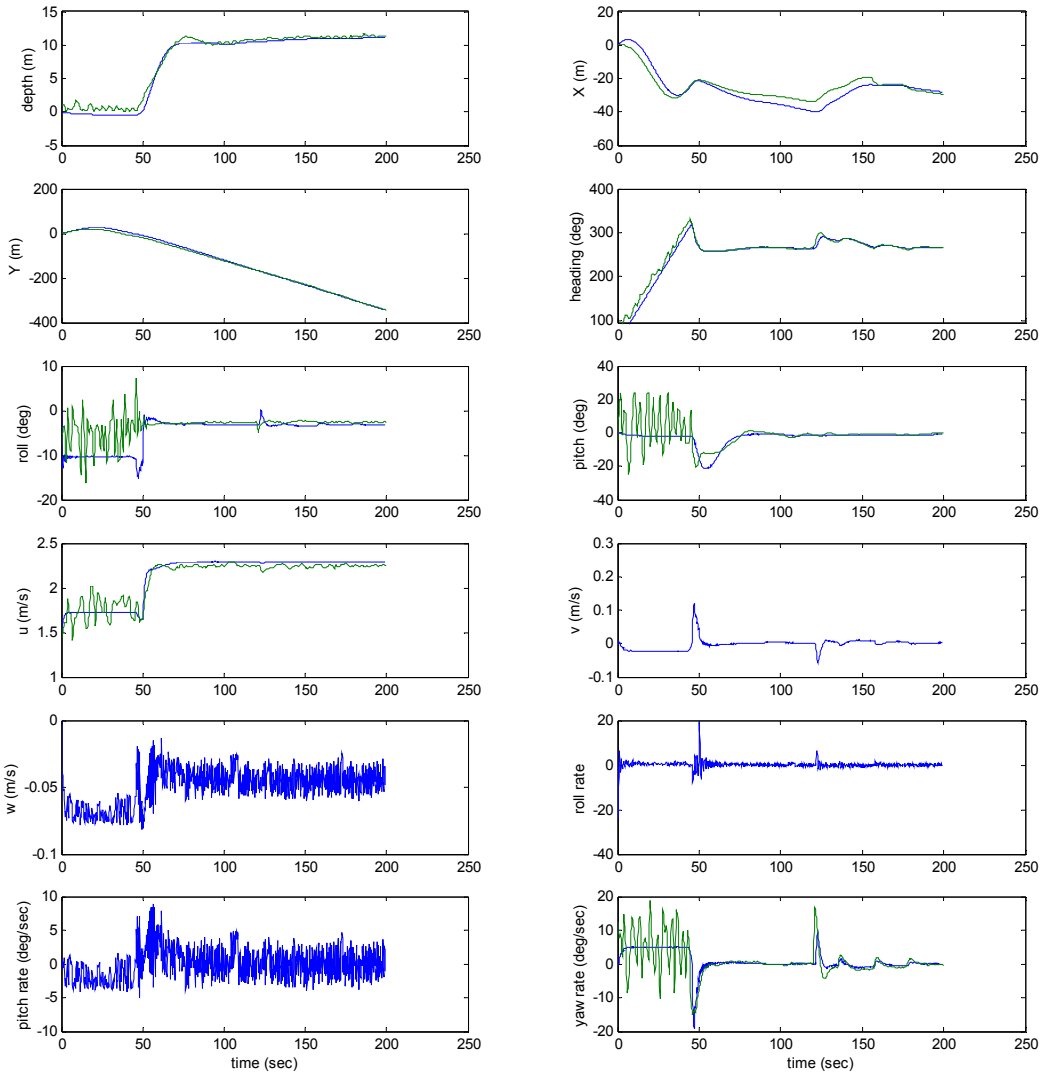

Figure 3.12. REMUS RUV simulation: state response (blue) to the depth and heading input commands shown in Figures 3.6. The experimentally measured responses (green) were available for selected state variables.

Chapter 4: RUV Sliding Mode Control

INTRODUCTION

This chapter discusses the development of the nonlinear sliding mode control for the RUV class. The first section outlines stability considerations for the RUV class with a equal or greater number of actuators than degrees of freedom. For this configuration it is easy to design a globally asymptotically stable control law. However, for the RUV class of this dissertation which has fewer actuators than degrees of freedom, application of these so-called *computed torque* techniques can induce unmatched disturbances. Unmatched disturbances further result from subsystem coupling terms and environmental perturbations such as wave induced forces (c.f. Chapter 6). After developing a few mathematical preliminaries, the control laws for the depth and heading subsystems are derived. The controllers are simulated with the full nonlinear simulation equations and REMUS parameters calculated in Chapter 3. Results are given and compared with the linear controllers developed in the final section of Chapter 3.

STABILITY CONSIDERATIONS FOR RUVS WITH INCONSISTENT ACTUATOR CONFIGURATIONS

Under the assumption of equal or greater number of actuators than degrees of freedom, [Fossen] shows that the nonlinear equations describing general six degree of freedom rigid body motion in a viscous medium can be globally asymptotically stabilized

in regulation and tracking by using the computed torque method (effectively feedback linearization). However, for the class of vehicles with an inconsistent actuator design where the number of actuators is few than the number of degrees of freedom, the results do not apply. To understand this, assume as [Fossen] does that the vector of control forces and torques, τ, is linearly related to the input signal, U, through the control input matrix, G, by

$$\tau = GU .$$

Let m be the number of control inputs (actuators) and n be the number of degrees of freedom. Now suppose that $\tau \in \mathbb{R}^{n\times 1}$, $G \in \mathbb{R}^{n\times m}$ and $U \in \mathbb{R}^{m\times 1}$ where $m < n$. Even though it is inconsistent, the equation in fact has a unique least squares solution, assuming G is full rank [Strang], given by the left pseudo-inverse

$$U_{LS} = \left(G^T G\right)^{-1} G^T \tau .$$

Figure 4.1 illustrates the situation for $n=3$, $m=2$ when the computed torque input, τ_{CT}, is not in the column space of G.

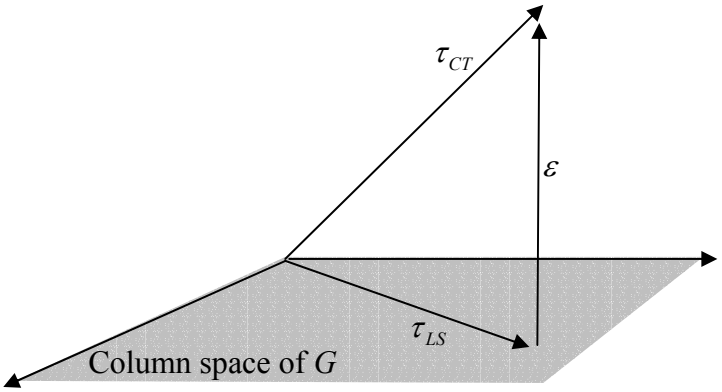

Figure 4.1. The relation between the globally stable controller, τ_{CT}, and its least squares approximation, τ_{LS}. The error vector, ε, acts as a disturbance on the system since it is perpendicular to the input space spanned by the columns of G, i.e., it is an unmatched disturbance.

If the least squares solution, $\tau_{LS} = GU_{LS}$, is applied to the system, the error, $\varepsilon = |\tau_{CT} - \tau_{LS}|$ acts as a disturbance to the dynamics. [Silpa-Anan and Zelinsky] have implemented the least squares solution in a computed torque controller for an RUV with *n=6, m=5*. The authors assume unmatched parameter uncertainty and disturbances in the roll subspace will be passively rejected due to the vehicle design which places the center of buoyancy above the center of gravity, creating a righting moment in the roll dimension. For the RUV class of this dissertation this is a poor assumption since the roll

righting moment is small and m=3 so that the columns of the input matrix span fewer dimensions.

The preceding ideas illustrate that the control signal cannot generically be determined by pseudo inversion of the input matrix when the actuator configuration consists of a set of forces and moments generated through the dependent interaction of thrusters and control surfaces. Researchers in the past have used methods which assume that the surge, depth and heading subsystems can be decoupled and a separate controller designed for each decoupled subsystem [Allen et. al. 1997], [Fossen].

The optimally-robust control of the specified RUV class can be achieved by developing sliding mode controllers with optimally chosen performance term gains for each of the separate subsystems in heading, depth and surge. However, the coupling terms and environmental effects must be included in each separate control law as unmodelled dynamics. Conventional sliding mode techniques which assume the subsystem can be transformed into the canonical *normal form* can only accommodate matched uncertainties and disturbances acting in the range of the input distribution (in linear systems theory this is analogous to the linear space spanned by the columns of the input matrix). Since the coupling terms and environmental perturbations can result in unmatched uncertainty and the subsystem equations cannot be rendered in normal form, a sliding mode approach is developed herein which allows some ability to stabilize the RUV and can accommodate a less approximate, nonlinear system representation. The resulting canonical representation is typically called the *regular form* [Isidori].

The following sections presents several mathematical preliminaries useful for the development of the controller. Control laws are then developed for the RUV class

heading and depth autopilots. Performance of the regular form sliding mode control is evaluated using the full nonlinear model developed in Chapter 3. The PD/PID heading and depth autopilots are used as a comparison baseline.

MATHEMATICAL PRELIMINARIES

This section details the mathematical preliminaries required to prove several theorems regarding the existence and stability of the regular form sliding mode control law. Consider the *single input* system given by

$$\dot{x} = f(x) + g(x) \cdot u \tag{4.1}$$

where $x \in \mathbb{R}^{n\times 1}$ is the state vector, $f \in \mathbb{R}^{n\times 1}$ a smooth (at least C^r with r smooth derivatives) vector field, $g \in \mathbb{R}^{n\times 1}$ is a smooth vector field, and $u \in \mathbb{R}$.

Definition 4.1. *Covector field*. A covector field is a smooth mapping assigning a point $x \in U \subseteq \mathbb{R}^n$ where U is an open set of $\mathbb{R}^n$ to a vector in $\left(\mathbb{R}^n\right)^{\perp}$, the set of all n-dimensional row vectors.

Definition 4.2. *Lie Bracket*. The Lie Bracket of two smooth n-vector fields is defined as

$$[f,g](x)=\frac{\partial g(x)}{\partial x}f(x)-\frac{\partial f(x)}{\partial x}g(x)\,. \tag{4.2}$$

Definition 4.3. *Distribution*. Suppose *d* smooth vector fields are given $f_1,\cdots,f_d$ defined $\forall x\in U\subseteq\mathbb{R}^n$ where U is an open set of $\mathbb{R}^n$. A smooth distribution is characterized by the span of the set of d smooth vector fields

$$\Delta=span\{f_1,\cdots,f_d\}\,. \tag{4.3}$$

Evaluated at each point $x\in U\subseteq\mathbb{R}^n$, a distribution is a vector space which is a subspace of $\mathbb{R}^n$

$$\Delta(x)=span\{f_1(x),\cdots,f_d(x)\}\,.$$

Definition 4.4. *Nonsingular Distribution.* A distribution is nonsingular if there exists an integer d such that

$$\dim(\Delta(x))=d$$

$\forall x\in U$.

Definition 4.5. *Annihilator.* Given a distribution Δ, for each $x\in U$, the annihilator distribution, $\Delta^{\perp}$, is the collection of all covectors that annihilate vectors in $\Delta(x)$

$$\Delta^{\perp} \equiv \left\{ w^{\perp} \in (\mathbb{R}^n)^{\perp} \mid w^{\perp} \cdot v = 0, \forall v \in \Delta(x) \right\}. \tag{4.4}$$

Definition 4.6. *Involutivity.* A distribution is involutive if for any two vector fields in a distribution, $\tau_1, \tau_2 \in \Delta$, their Lie Bracket (Definition 4.2) is also in the distribution, $\left[\tau_1, \tau_2\right] \in \Delta$.

Definition 4.7. *Integrability.* A nonsingular d-dimensional distribution Δ defined on some open set U of $\mathbb{R}^n$ is completely integrable if for each $\underline{x}$ of U, there is a neighborhood $\underline{U}$ of $\underline{x}$ and n-d smooth functions $\lambda_1, \cdots, \lambda_{n-d}$ defined on $\underline{U}$, the annihilator of Δ is spanned by a set of n-d exact differentials

$$span\left\{ d\lambda_1, \cdots, d\lambda_{n-d} \right\} = \Delta^{\perp}. \tag{4.5}$$

Definition 4.8. *Diffeomorphism.* A diffeomorphism is a change of coordinates defined by

$$\varsigma = \Phi(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_n(x) \end{bmatrix} \tag{4.6}$$

where (i) $\Phi(x)$ is invertible and (ii) $\Phi(x)$ and $\Phi^{-1}(\varsigma)$ are smooth mappings $\forall x \in U \subseteq \mathbb{R}^n$. If $U = \mathbb{R}^n$, the diffeomorphism is *global*.

Definition 4.9. *Regular Form.* A single input system is in regular form if there exist smooth vector fields $f_I \in \mathbb{R}^{n-1}$, $f_{II} \in \mathbb{R}^1$, and $g_{II} \in \mathbb{R}^1$ and a set of partitioned states $\zeta_I \in \mathbb{R}^{n-1}$ and $\zeta_{II} \in \mathbb{R}^1$ such that

$$\begin{aligned} \dot{\zeta}_I &= f_I(\zeta_I, \zeta_{II}) \\ \dot{\zeta}_{II} &= f_{II}(\zeta_I, \zeta_{II}) + g_{II}(\zeta_I, \zeta_{II}) \cdot u. \end{aligned} \tag{4.7}$$

Theorem 4.1. *Frobenius.* A nonsingular distribution is completely integrable if and only if it is involutive.

Proof. The proof can be found in [Isidori].

Theorem 4.2. *Existence of a Regular Form (single input system).* A diffeomorphism $\varsigma = \Phi(x)$, defined $\forall x \in U \subseteq \mathbb{R}^n$, exists which transforms the single input system of Equation 4.1 into regular form (Definition 4.9) if there exists a nonsingular, involutive distribution Δ satisfying

$$span\{g\} \subseteq \Delta \tag{4.8}$$

for a smooth input vector field $g \in \mathbb{R}^n$ which is nonzero in some neighborhood U of $\mathbb{R}^n$.

Proof. The proof is constructive. Let $\Delta = span\{g\}$. Since the smooth input vector field $g \in \mathbb{R}^n$ is nonzero in some neighborhood U of $\mathbb{R}^n$, the dimension of Δ is unity in that neighborhood and therefore Δ is nonsingular in U by Definition 4.4. Furthermore since Δ is spanned by a single vector *g*, then

$$[g, g](x) = \frac{\partial g(x)}{\partial x} g(x) - \frac{\partial g(x)}{\partial x} g(x) = 0$$

Because $\Delta(x)$ is a vector space, it includes the zero vector. Therefore Δ is involutive by Definition 4.4 since $g \in \Delta \Rightarrow [g, g] \in \Delta$. By Theorem 4.1, since the distribution Δ is nonsingular and involutive, it is integrable. Therefore the annihilator $\Delta^{\perp}$ is spanned by a set of n-1 exact differentials

$$span\{d\lambda_1, \cdots, d\lambda_{n-1}\} = \Delta^{\perp}.$$

One can solve for a set of functions $\{\lambda_1, \cdots, \lambda_{n-1}\}$ since by definition of the annihilator the λ_i's satisfy

$$\frac{\partial \lambda_j}{\partial x} \cdot g = \frac{\partial \lambda_j}{\partial x_1} \cdot g_1 + \cdots \frac{\partial \lambda_j}{\partial x_n} \cdot g_n = 0, \quad j = 1 : n-1. \tag{4.9}$$

Form the required diffeomorphism as

$$\varsigma = \Phi(x) = \begin{bmatrix} \lambda_1(x) \\ \vdots \\ \lambda_{n-1}(x) \\ x_j \end{bmatrix}$$

where x_j is arbitrarily chosen $1 \le j \le n$. To show that the transformation leads to the regular form (Definition 4.9), take the first time derivative:

$$\dot{\varsigma} = \frac{\partial \Phi(x)}{\partial x} \dot{x} = \begin{bmatrix} \frac{\partial \lambda_1(x)}{\partial x} \dot{x} \\ \vdots \\ \frac{\partial \lambda_{n-1}(x)}{\partial x} \dot{x} \\ \dot{x}_j \end{bmatrix} = \begin{bmatrix} \frac{\partial \lambda_1(x)}{\partial x} (f + gu) \\ \vdots \\ \frac{\partial \lambda_{n-1}(x)}{\partial x} (f + gu) \\ f_j + g_j u \end{bmatrix} = \begin{bmatrix} \frac{\partial \lambda_1(x)}{\partial x} f + \frac{\partial \lambda_1(x)}{\partial x} gu \\ \vdots \\ \frac{\partial \lambda_{n-1}(x)}{\partial x} f + \frac{\partial \lambda_{n-1}(x)}{\partial x} gu \\ f_j + g_j u \end{bmatrix}.$$

Noting Equation 4.9, this reduces to

$$\dot{\varsigma} = \begin{bmatrix} \frac{\partial \lambda_1(x)}{\partial x} \cdot f \\ \vdots \\ \frac{\partial \lambda_{n-1}(x)}{\partial x} \cdot f \\ f_j + g_j u \end{bmatrix} = \begin{bmatrix} \frac{\partial \lambda_1(x)}{\partial x} \cdot f \\ \vdots \\ \frac{\partial \lambda_{n-1}(x)}{\partial x} \cdot f \\ \hline f_j \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \hline g_j u \end{bmatrix} \equiv \begin{bmatrix} f_I(\zeta_I, \zeta_{II}) \\ \hline f_{II}(\zeta_I, \zeta_{II}) \end{bmatrix} + \begin{bmatrix} 0_{(n-1)x1} \\ \hline g_{II}(\zeta_I, \zeta_{II}) \end{bmatrix} u$$

where f_I, f_{II} and g_{II} are defined in the obvious way from the column vector partition. □

Note that for the RUV model of Chapter 3 the distribution $\Delta = span\{g\}$ will be singular at zero surge speed and therefore a regular form will not exist unless the vehicle is moving.

REGULAR FORM SLIDING MODE CONTROL

In this section the existence of the regular form transformation for the case of a single input system is proven. Previous approaches to control of robotic underwater vehicles involve using decoupled PD/PID autopilots [Allen, et. al., 1997] Single-Input/Multi-state [Cristi, et. al.], [Fossen], as well as techniques that make dynamic and kinematic assumptions which lead to a normal form for the heading and depth control subsystems [Yoerger and Slotine]. In this section the regular form sliding mode control algorithm is derived. The advantage of such a regular form approach includes increased generality in the class of systems to which techniques can be applied. Furthermore, as will be shown, one can include more states in the subsystem control law than can be

assumed under normal form conditions. This leads to increased control performance and robustness.

Let the sliding surface be defined as

$$\sigma(\zeta) = -s^T \zeta_I + \zeta_{II} \equiv \overline{s}^T \zeta \,. \tag{4.10}$$

Theorem 4.3. There exists a real number η such that the control $u = u_p + u_{eq}$ where

$$u_p = -\frac{\eta}{g_{II}} sign(\sigma(\zeta)), \quad u_{eq} = -\frac{\overline{s}^T \begin{bmatrix} \hat{f}_I(\zeta) \\ \hat{f}_{II}(\zeta) \end{bmatrix}}{g_{II}}, \tag{4.11}$$

asymptotically stabilizes the system of Equation 4.7 to the (*n-1*)-dimensional manifold $\sigma(\varsigma) = 0$.

Proof. Let a pseudo-Lyapunov function candidate be $V = \frac{1}{2}\sigma^2$. Clearly $V > 0, \forall \sigma \neq 0$. The time derivative of V is

$$
\begin{aligned}
\dot{V} &= \sigma \overline{s}^T \dot{\zeta} \\
&= \sigma \overline{s}^T \left(\begin{bmatrix} f_I(\zeta) \\ f_{II}(\zeta) \end{bmatrix} + \begin{bmatrix} 0 \\ g_{II} \end{bmatrix} u \right) \\
&= \sigma \left(\overline{s}^T \begin{bmatrix} f_I(\zeta) \\ f_{II}(\zeta) \end{bmatrix} - \overline{s}^T \begin{bmatrix} 0 \\ g_{II} \end{bmatrix} \left(\frac{\eta}{g_{II}} sign(\sigma) + \frac{\overline{s}^T \begin{bmatrix} \hat{f}_I(z) \\ \hat{f}_{II}(\zeta) \end{bmatrix}}{g_{II}} \right) \right) \\
&= \sigma \left(\overline{s}^T \begin{bmatrix} \Delta f_I(\zeta) \\ \Delta f_{II}(\zeta) \end{bmatrix} - \eta \cdot sign(\sigma) \right).
\end{aligned}
\tag{4.12}
$$

If the so-called switching gain η is chosen such that

$$
\eta = \left\| \overline{s} \right\| \left\| \begin{bmatrix} \Delta f_I \\ \Delta f_{II} \end{bmatrix} \right\| + \varepsilon, \quad \varepsilon > 0 \tag{4.13}
$$

then from (4.12)

$$
\begin{aligned}
\dot{V} &= \overline{s}^T \begin{bmatrix} \Delta f_I(\zeta) \\ \Delta f_{II}(\zeta) \end{bmatrix} \sigma - \eta \cdot sign(\sigma) \cdot \sigma \\
&= \overline{s}^T \begin{bmatrix} \Delta f_I(\zeta) \\ \Delta f_{II}(\zeta) \end{bmatrix} \sigma - \eta \cdot |\sigma| \\
&= \overline{s}^T \begin{bmatrix} \Delta f_I(\zeta) \\ \Delta f_{II}(\zeta) \end{bmatrix} \sigma - \left\| \overline{s} \right\| \left\| \begin{bmatrix} \Delta f_I \\ \Delta f_{II} \end{bmatrix} \right\| |\sigma| - \varepsilon \cdot |\sigma| \\
&< \left\| \overline{s} \right\| \left\| \begin{bmatrix} \Delta f_I \\ \Delta f_{II} \end{bmatrix} \right\| |\sigma| - \left\| \overline{s} \right\| \left\| \begin{bmatrix} \Delta f_I \\ \Delta f_{II} \end{bmatrix} \right\| |\sigma| - \varepsilon \cdot |\sigma| \\
&< -\varepsilon \cdot |\sigma|
\end{aligned}
\tag{4.14}
$$

By Lyapunov's stability theorem, $\sigma \to 0$ as $t \to \infty$. □

Incidentally, the performance term in Equation 4.11 can be modified to include a factor δ in the denominator of the *sign* term to provide a boundary layer [Slotine and Li]:

$$u_p = -\frac{\eta}{g_{II}} sign(\frac{\sigma(\zeta)}{\delta}) . \tag{4.15}$$

This boundary acts to "dampen" the switching response of the controller. As will be shown in Chapter 5, a real implementation can substantially improve the dynamic response by increasing δ since this effectively reduces the activity level of a switched control law.

Since the system can be made to reach a sliding mode, the following three conditions are asserted:

Condition 1. $\sigma = 0$

Condition 2. $\dot{\sigma} = 0$

Condition 3. $u = u_{eq}$

From **Condition 1**, one has

$$\sigma(\zeta) = \overline{s}^T \begin{bmatrix} \zeta_I \\ \zeta_{II} \end{bmatrix} = -s^T \zeta_I + \zeta_{II} = 0 \tag{4.16}$$

Therefore $\zeta_{II} = s^T \zeta_I$ which shows that in sliding mode

$$\dot{\zeta}_I = f_I(\zeta_I, s^T \zeta_I) \tag{4.17}$$

and the sliding surface parameter vector, *s*, can be chosen to stabilize the so-called reduced order dynamic system, ζ_I [Perruquetti, et. al.].

APPLICATION TO THE RUV MODEL

In this section the regular form transformation theorem is applied to the RUV model developed in the Chapter 3 in order to transform the heading and depth subsystems and apply the sliding mode control technique.

Heading Sliding Mode Controller Autopilot

Following the development in the *State Linearized Control Design* sections of Chapter 3, assume the heading-related states can be can be decoupled from the system by defining the heading substate $x_h = \begin{bmatrix} v & r & \psi \end{bmatrix}^T$ and nominal states

$$\begin{bmatrix} \nu_0 \\ \eta_0 \end{bmatrix} = \begin{bmatrix} u_0 & v & 0 & 0 & 0 & r & 0 & 0 & 0 & 0 & 0 & \psi \end{bmatrix}^T. \tag{4.18}$$

Then the controller design equations are

$$\dot{x}_h = f_h(x_h) + g_h \delta_r \tag{4.19}$$

where

$$f_h = M_h^{-1} \cdot \begin{bmatrix} -m(u_0 r - y_g r^2) + Y_{HS} + Y_{v|v|} v|v| + Y_{r|r|} r|r| + Y_{uv} u_0 v \\ -m(x_g u_0 r + y_g v r) + N_{HS} + N_{v|v|} v|v| + N_{r|r|} r|r| + N_{uv} u_0 v + Y_{ur} u_0 r \\ r \end{bmatrix}, \tag{4.20}$$

$$g_h = M_h^{-1} \cdot \begin{bmatrix} Y_{uu\delta_r} u_0^2 \\ N_{uu\delta_r} u_0^2 \\ 0 \end{bmatrix}, \tag{4.21}$$

and

$$M_h = \begin{bmatrix} m - Y_{\dot{v}} & m x_g - Y_{\dot{r}} & 0 \\ m x_g - N_{\dot{v}} & I_z - N_{\dot{r}} & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{4.22}$$

and δ_r is the rudder angle. Making the further simplification of ignoring the off diagonal terms in M_h allows us to decouple the equations. This is justified by the fact that the off-diagonal terms are small compared to the diagonal values in M_h. The equations for f_h and g_h become

$$f_h = \begin{bmatrix} \dfrac{1}{m - Y_{\dot{v}}}\left\{-m(u_0 r - y_g r^2) + Y_{HS} + Y_{v|v|} v|v| + Y_{r|r|} r|r| + Y_{uv} u_0 v\right\} \\ \dfrac{1}{I_z - N_{\dot{r}}}\left\{-m(x_g u_0 r + y_g v r) + N_{HS} + N_{v|v|} v|v| + N_{r|r|} r|r| + N_{uv} u_0 v + + Y_{ur} u_0 r\right\} \\ r \end{bmatrix} \quad (4.23)$$

and

$$g_h = \begin{bmatrix} \dfrac{1}{m - Y_{\dot{v}}}\left\{Y_{uu\delta_r} u_0^2\right\} \\ \dfrac{1}{I_z - N_{\dot{r}}}\left\{N_{uu\delta_r} u_0^2\right\} \\ 0 \end{bmatrix} \quad (4.24)$$

respectively.

The next step is to check the conditions of the Regular Form Theorem (Theorem 4.2). Let $\Delta_h = span\{g_h\}$. Then

$$\Delta_h = span\left\{\begin{bmatrix} Y_{uu\delta_r} u_0^2 \\ N_{uu\delta_r} u_0^2 \\ 0 \end{bmatrix}\right\} \quad (4.25)$$

and as such is nonsingular $\forall u_0 \neq 0$. Furthermore every distribution spanned by a single smooth vector field is involutive, since clearly $\tau \in \Delta \Rightarrow [\tau, \tau] = 0 \in \Delta$. Therefore the theorem asserts the existence of a regular coordinate transformation. The proof of the

theorem illustrates the method of calculation, viz., let $\Delta_h = span\{g_h\}$ and find a basis for the annihilator distribution $\Delta_h^{\perp}$:

$$\begin{aligned} 0 &= d\lambda_h \cdot g \\ &= \frac{\partial \lambda_h}{\partial x_{h1}} g_{h1} + \frac{\partial \lambda_h}{\partial x_{h2}} g_{h2} \\ &= \frac{\partial \lambda_h}{\partial x_{h1}} \frac{1}{m - Y_{\dot{v}}} \left\{ Y_{uu\delta_r} u_0^2 \right\} + \frac{\partial \lambda_h}{\partial x_{h2}} \frac{1}{I_z - N_{\dot{r}}} \left\{ N_{uu\delta_r} u_0^2 \right\} \end{aligned} \tag{4.26}$$

This is satisfied by

$$\lambda_h = \frac{1}{I_z - N_{\dot{r}}} N_{uu\delta_r} u_0^2 \cdot v - \frac{1}{m - Y_{\dot{v}}} Y_{uu\delta_r} u_0^2 \cdot r \tag{4.27}$$

The coordinate transform is therefore

$$\begin{bmatrix} \zeta_{h1} \\ \zeta_{h2} \\ \zeta_{h3} \end{bmatrix} = \Phi_h(x_h) = \begin{bmatrix} \psi \\ \dfrac{1}{I_z - N_{\dot{r}}} \left\{ N_{uu\delta_r} u_0^2 \right\} v - \dfrac{1}{m - Y_{\dot{v}}} \left\{ Y_{uu\delta_r} u_0^2 \right\} r \\ r \end{bmatrix} \tag{4.28}$$

To simplify notation, let α_h and β_h be defined as

$$\alpha_h = \frac{1}{m - Y_{\dot{v}}}\left\{Y_{uu\delta_r} u_0^2\right\}$$
$$\beta_h = \frac{1}{I_z - N_{\dot{r}}}\left\{N_{uu\delta_r} u_0^2\right\} \tag{4.29}$$

Then equations of motion in the transformed coordinates are

$$\dot{\zeta}_{h1} = f_{\zeta h1} = \zeta_{h3}$$

$$\dot{\zeta}_{h2} = f_{\zeta h2} = \frac{u_0^2}{(m - Y_{\dot{v}})(I_z - N_{\dot{r}})}\left\{\begin{array}{l} N_{uu\delta_r}\left\{-m\left(u_0\zeta_{h3} - y_g\zeta_{h3}^2\right) + Y_{HS} + Y_{v|v|}\left(\frac{1}{\beta_h}\zeta_{h2} + \frac{\alpha_h}{\beta_h}\zeta_{h3}\right)\left|\frac{1}{\beta_h}\zeta_{h2} + \frac{\alpha_h}{\beta_h}\zeta_{h3}\right| + Y_{ur}u_0\zeta_{h3} + Y_{uv}u_0\left(\frac{1}{\beta_h}\zeta_{h2} + \frac{\alpha_h}{\beta_h}\zeta_{h3}\right)\right\} \\ -Y_{uu\delta_r}\left\{\begin{array}{l} -m\left(x_g u_0\zeta_{h3} + y_g\zeta_{h3}\left(\frac{1}{\beta_h}\zeta_{h2} + \frac{\alpha_h}{\beta_h}\zeta_{h3}\right)\right) + N_{HS} + N_{v|v|}\left(\frac{1}{\beta_h}\zeta_{h2} + \frac{\alpha_h}{\beta_h}\zeta_{h3}\right)\left|\frac{1}{\beta_h}\zeta_{h2} + \frac{\alpha_h}{\beta_h}\zeta_{h3}\right| \\ +N_{r|r|}\zeta_{h3}\left|\zeta_{h3}\right| + N_{uv}u_0\left(\frac{1}{\beta_h}\zeta_{h2} + \frac{\alpha_h}{\beta_h}\zeta_{h3}\right) + Y_r u_0\zeta_{h3} \end{array}\right. \end{array}\right\}$$

$$\dot{\zeta}_{h3} = f_{\zeta h3} + g_{\zeta h}\delta_r = \frac{1}{I_z - N_{\dot{r}}}\left\{\begin{array}{l} -m\left(x_g u_0\zeta_{h3} + y_g\zeta_{h3}\left(\frac{1}{\beta_h}\zeta_{h2} + \frac{\alpha_h}{\beta_h}\zeta_{h3}\right)\right) + N_{HS} + N_{v|v|}\left(\frac{1}{\beta_h}\zeta_{h2} + \frac{\alpha_h}{\beta_h}\zeta_{h3}\right)\left|\frac{1}{\beta_h}\zeta_{h2} + \frac{\alpha_h}{\beta_h}\zeta_{h3}\right| \\ +N_{r|r|}\zeta_{h3}\left|\zeta_{h3}\right| + N_{uv}u_0\left(\frac{1}{\beta}\zeta_2 + \frac{\alpha_h}{\beta_h}\zeta_{h3}\right) + Y_r u_0\zeta_{h3} \end{array}\right\} + \left(\frac{1}{I_z - N_{\dot{r}}}N_{uu\delta_r}u_0^2\right)\delta_r \tag{4.30}$$

Note that δ_r appears only in the last equation. As a notational convenience here and in what follows define $f_{I,h}$ and $f_{II,h}$ by partitioning the above appropriately

$$\left[\begin{array}{c} \dot{\zeta}_{I,h} \\ \hline \dot{\zeta}_{II,h} \end{array}\right] = f_{\zeta h} + g_{\zeta h}\delta_r = \left[\begin{array}{c} f_{I,h} \\ \hline f_{II,h} \end{array}\right] + \left[\begin{array}{c} 0_{2x1} \\ \hline g_{II,h} \end{array}\right]\delta_r \tag{4.31}$$

where

$$\left[\frac{f_{I,h}}{f_{II,h}}\right] \equiv \left[\begin{array}{c} f_{\zeta h1} \\ \hdashline f_{\zeta h2} \\ \hdashline f_{\zeta h3} \end{array}\right], \quad g_{II,h} \equiv g_{\zeta h}, \tag{4.32}$$

and the individual components are defined according to Equation 4.30 above.

In order to transform the control law into vehicle coordinates take the time derivative of $\zeta = \Phi(x)$. This yields

$$\begin{aligned} \dot{\zeta} &= \frac{\partial \Phi}{\partial x}\dot{x} \\ &= f_{\zeta h} + g_{\zeta h}\delta_r \end{aligned} \quad . \tag{4.33}$$

Therefore

$$\dot{x} = \left(\frac{\partial \Phi}{\partial x}\right)^{-1}\left(f_{\zeta h} + g_{\zeta h}\delta_r\right) = \left(\frac{\partial \Phi}{\partial x}\right)^{-1} f_{\zeta h}\left(\Phi(x)\right) + \left(\frac{\partial \Phi}{\partial x}\right)^{-1} g_{\zeta h}\left(\Phi(x)\right)\cdot\delta_r\left(\Phi(x)\right). \tag{4.34}$$

Since

$$f_h(x_h) = \left(\frac{\partial \Phi}{\partial x}\right)^{-1} f_{\zeta h}\left(\Phi(x)\right), \quad \textit{and} \quad g_h(x_h) = \left(\frac{\partial \Phi}{\partial x}\right)^{-1} g_{\zeta h}\left(\Phi(x)\right) \tag{4.35}$$

the control law can be derived in terms of the x_h coordinates in order to implement the regular form sliding mode controller heading autopilot system. This is done in order to avoid implementing an additional set of state equations (Equation 4.30). Thus the control signal is

$$\begin{aligned}\delta_r\left(\Phi_h\left(x_h\right)\right) &= -\frac{\eta_{\Delta f_{\zeta h}}}{\overline{s}_h^{\,T} g_{\zeta h}} sign(\sigma_h(\Phi_h(x_h))) - \frac{\overline{s}_h^{\,T}}{\overline{s}_h^{\,T} g_{\zeta h}}(\hat{f}_{\zeta h} - \dot{\zeta}_{hD}) \\ &= -\frac{\eta_{\Delta f_{\zeta h}}}{\overline{s}_h^{\,T} \dfrac{\partial \Phi_h}{\partial x_h} g_h} sign(\sigma_h(\Phi_h(x_h))) - \frac{\overline{s}_h^{T}}{\overline{s}_h^{\,T} \dfrac{\partial \Phi_h}{\partial x_h} g_h} \frac{\partial \Phi_h}{\partial x_h}(\hat{f}_h - \dot{x}_{hD})\end{aligned} \tag{4.36}$$

where, from Theorem 4.3, the performance term gain is chosen such that

$$\eta_{\Delta f_{\zeta h}} > \left\|\overline{s}_h\right\| \left\| \begin{matrix} \Delta f_{\zeta h1} \\ \Delta f_{\zeta h2} \\ \Delta f_{\zeta h3} \end{matrix} \right\| = \left\|\overline{s}_h\right\| \left\| \frac{\partial \Phi_h}{\partial x_h} \right\| \left\| \begin{matrix} \Delta f_{h1} \\ \Delta f_{h2} \\ \Delta f_{h3} \end{matrix} \right\|.$$

Depth Sliding Mode Controller Autopilot

The development of the regular form sliding mode depth autopilot control law follows the same steps. Defining the depth substate $x_d = \begin{bmatrix} w & q & z & \theta \end{bmatrix}^T$ and nominal states

$$\begin{bmatrix} \nu_0 \\ \eta_0 \end{bmatrix} = \begin{bmatrix} u_0 & 0 & w & 0 & q & 0 & 0 & 0 & z & 0 & \theta & 0 \end{bmatrix}^T . \tag{4.37}$$

The controller design equations are

$$\dot{x}_d = f_d(x_d) + g_d \delta_s \tag{4.38}$$

where

$$f_d = M_d^{-1} \cdot \begin{bmatrix} m(u_0 q + z_g q^2) + Z_{HS} + Z_{w|w|} w|w| + Z_{q|q|} q|q| + Z_{uq} u_0 q + Z_{uw} u_0 w \\ -m(x_g u_0 q + z_g w q) + M_{HS} + M_{w|w|} w|w| + M_{q|q|} q|q| + M_{uq} u_0 q + M_{uw} u_0 w \\ -u_0 \sin\theta + w\cos\theta \\ q \end{bmatrix}, \tag{4.39}$$

$$g_d = M_d^{-1} \cdot \begin{bmatrix} Z_{uu\delta_s} u_0^2 \\ M_{uu\delta_s} u_0^2 \\ 0 \\ 0 \end{bmatrix}, \tag{4.40}$$

and

$$M_d = \begin{bmatrix} m - Z_{\dot{w}} & 0 & 0 & 0 \\ 0 & I_y - M_{\dot{q}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.41}$$

and δ_s is the stern angle input. Similar to the heading autopilot, one can make the simplifying assumption of zero off-diagonal inertia matrix elements justified by the fact that the off diagonal terms are small compared to the diagonal values in M_d. The equations for f_d and g_d become

$$f_d = \begin{bmatrix} \frac{1}{m - Z_{\dot{w}}}\left\{m(u_0 q + z_g q^2) + Z_{HS} + Z_{w|w|} w|w| + Z_{q|q|} q|q| + Z_{uq} u_0 q + Z_{uw} u_0 w\right\} \\ \frac{1}{I_y - M_{\dot{q}}}\left\{-m(x_g u_0 q + z_g w q) + M_{HS} + M_{w|w|} w|w| + M_{q|q|} q|q| + M_{uq} u_0 q + M_{uw} u_0 w\right\} \\ -u_0 \sin\theta + w\cos\theta \\ q \end{bmatrix} \tag{4.42}$$

and

$$g_d = \begin{bmatrix} \frac{1}{m - Z_{\dot{w}}}\left\{Z_{uu\delta_s} u_0^2\right\} \\ \frac{1}{I_y - M_{\dot{q}}}\left\{M_{uu\delta_s} u_0^2\right\} \\ 0 \\ 0 \end{bmatrix}, \tag{4.43}$$

respectively.

Since g_d is constant and nonzero everywhere except u_0 = 0, the distribution defined as $\Delta_d = span\{g_d\}$ is nonsingular $\forall u_0 \neq 0$ and involutive. Therefore Theorem 4.2 ensures that a regular coordinate transformation exists for the depth subsystem

coordinates. Using the approach found in the heading autopilot section for establishing a basis for the annihilator distribution $\Delta_d^{\perp}$ it is seen that

$$\begin{aligned} 0 &= d\lambda_d \cdot g \\ &= \frac{\partial \lambda_d}{\partial x_{d1}} g_{d1} + \frac{\partial \lambda_d}{\partial x_{d2}} g_{d2} \\ &= \frac{\partial \lambda_d}{\partial x_{d1}} \frac{1}{m - Z_{\dot{w}}} \left\{ Z_{uu\delta_s} u_0^2 \right\} + \frac{\partial \lambda_d}{\partial x_{d2}} \frac{1}{I_y - M_{\dot{q}}} \left\{ M_{uu\delta_s} u_0^2 \right\} \end{aligned} \tag{4.44}$$

is satisfied by

$$\begin{aligned} \lambda_d &= \frac{1}{I_y - M_{\dot{q}}} M_{uu\delta_s} u_0^2 \cdot w - \frac{1}{m - Z_{\dot{w}}} Z_{uu\delta_s} u_0^2 \cdot q \\ &\equiv \beta_d w - \alpha_d q \end{aligned} \tag{4.45}$$

with α_d and β_d defined (similarly) in the obvious way. The coordinate transform is therefore

$$\begin{bmatrix} \zeta_{d1} \\ \zeta_{d2} \\ \zeta_{d3} \\ \zeta_{d4} \end{bmatrix} = \Phi_d \left(x_d \right) = \begin{bmatrix} z \\ \theta \\ \dfrac{1}{I_z - N_{\dot{r}}} \left\{ N_{uu\delta_r} u_0^2 \right\} v - \dfrac{1}{m - Y_{\dot{v}}} \left\{ Y_{uu\delta_r} u_0^2 \right\} r \\ q \end{bmatrix}. \tag{4.46}$$

Then depth subsystem equations of motion in the transformed coordinates are

$$\dot{\zeta}_{d1} = f_{\zeta d1} = -u_0 \sin\zeta_{d2} + \left(\frac{1}{\beta_d}\zeta_{d3} + \frac{\alpha_d}{\beta_d}\zeta_{d4}\right)\cos\zeta_{d2}$$

$$\dot{\zeta}_{d2} = f_{\zeta d2} = \zeta_{d4}$$

$$\dot{\zeta}_{d3} = f_{\zeta d3} = \frac{u_0^2}{(m - Z_{\dot{w}})(I_y - M_{\dot{q}})}\left\{\begin{array}{l} M_{uu\delta_s}\left\{\begin{array}{l} m\left(u_0\zeta_{d4} + z_g\zeta_{d4}^2\right) + Z_{HS} + Z_{w|w|}\left(\frac{1}{\beta_d}\zeta_{d3} + \frac{\alpha_d}{\beta_d}\zeta_{d4}\right)\left|\frac{1}{\beta_d}\zeta_{d3} + \frac{\alpha_d}{\beta_d}\zeta_{d4}\right| + Z_{q|q|}\zeta_{d4}\left|\zeta_{d4}\right| + Z_{uq}u_0\zeta_{d4} \\ + Z_{uw}u_0\left(\frac{1}{\beta_d}\zeta_{d3} + \frac{\alpha_d}{\beta_d}\zeta_{d4}\right) \end{array}\right. \\ -Z_{uu\delta_s}\left\{\begin{array}{l} -m\left(x_g u_0\zeta_{d4} + z_g\zeta_{d4}\left(\frac{1}{\beta_d}\zeta_{d3} + \frac{\alpha_d}{\beta_d}\zeta_{d4}\right)\right) + M_{HS} + M_{w|w|}\left(\frac{1}{\beta_d}\zeta_{d3} + \frac{\alpha_d}{\beta_d}\zeta_{d4}\right)\left|\frac{1}{\beta_d}\zeta_{d3} + \frac{\alpha_d}{\beta_d}\zeta_{d4}\right| \\ + M_{q|q|}\zeta_{d4}\left|\zeta_{d4}\right| + M_{uq}u_0\zeta_{d4} + M_{uw}u_0\left(\frac{1}{\beta_d}\zeta_{d3} + \frac{\alpha_d}{\beta_d}\zeta_{d4}\right) \end{array}\right. \end{array}\right\}$$

$$\dot{\zeta}_{d4} = f_{\zeta d4} + g_{\zeta d}\delta_s = \frac{1}{I_y - M_{\dot{q}}}\left\{\begin{array}{l} -m\left(x_g u_0\zeta_{d4} + z_g\zeta_{d4}\left(\frac{1}{\beta_d}\zeta_{d3} + \frac{\alpha_d}{\beta_d}\zeta_{d4}\right)\right) + M_{HS} + M_{w|w|}\left(\frac{1}{\beta_d}\zeta_{d3} + \frac{\alpha_d}{\beta_d}\zeta_{d4}\right)\left|\frac{1}{\beta_d}\zeta_{d3} + \frac{\alpha_d}{\beta_d}\zeta_{d4}\right| \\ + M_{q|q|}\zeta_{d4}\left|\zeta_{d4}\right| + M_{uq}u_0\zeta_{d4} + M_{uw}u_0\left(\frac{1}{\beta_d}\zeta_{d3} + \frac{\alpha_d}{\beta_d}\zeta_{d4}\right) \end{array}\right\} + \left(\frac{1}{I_y - M_{\dot{q}}}M_{uu\delta_s}u_0^2\right)\delta_s \tag{4.47}$$

Again, the control input appears only the last derivative. Define a similar partition as Equation 4.31

$$\left[\begin{array}{c} \dot{\zeta}_{I,d} \\ \hline \dot{\zeta}_{II,d} \end{array}\right] = f_{\zeta d} + g_{\zeta d}\delta_s = \left[\begin{array}{c} f_{I,d} \\ \hline f_{II,d} \end{array}\right] + \left[\begin{array}{c} 0_{3x1} \\ \hline g_{II,d} \end{array}\right]\delta_s \tag{4.48}$$

where again

$$\begin{bmatrix} f_{I,d} \\ \hdashline f_{II,d} \end{bmatrix} \equiv \begin{bmatrix} f_{\zeta d1} \\ f_{\zeta d2} \\ \hdashline f_{\zeta d3} \\ f_{\zeta d4} \end{bmatrix}, \quad g_{II,d} \equiv g_{\zeta d}\,, \tag{4.49}$$

is defined and the individual components are defined according to Equation 4.47 above.

Following the development of Equations (4.33) – (4.35) one obtains the depth control law in terms of the RUV frame variables

$$\begin{aligned} \delta_s\left(\Phi_d\left(x_d\right)\right) &= -\frac{\eta_{\Delta f_{\zeta d}}}{\overline{s}_d^{\,T} g_{\zeta d}} sign(\sigma_d(\Phi_d(x_d))) - \frac{\overline{s}^T}{\overline{s}^T g_{\zeta d}}(\hat{f}_{\zeta d} - \dot{\zeta}_{dD}) \\ &= -\frac{\eta_{\Delta f_{\zeta d}}}{\overline{s}_d^{\,T} \dfrac{\partial \Phi_d}{\partial x_d} g_d} sign(\sigma_d(\Phi_d(x_d))) - \frac{\overline{s}_d^{\,T}}{\overline{s}_d^{\,T} \dfrac{\partial \Phi_d}{\partial x_d} g_d} \frac{\partial \Phi_d}{\partial x_d}(\hat{f}_d - \dot{x}_{dD}) \end{aligned} \tag{4.50}$$

REGULAR FORM SLIDING MODE CONTROL RESULTS

This section presents the results of the closed-loop simulation of the foregoing control design on the robot vehicle model of Chapter 3. The cases of regulation, tracking, and robustness to parameter variation are examined. The latter performance is compared to the linearized controller developed in Chapter 3. In each case the combined heading and depth nonlinear regular form controllers are implemented on the full nonlinear model.

Controller Verification: Regulation

Figure 4.2 shows the regulation performance of the regular form sliding mode control law. The system is initialized at a depth of 0 meters and a heading of 90°. The desired depth and heading set-points are the same set-point values used in model verification development of Chapter 3. The results show that the sliding mode controller provide good performance reaching the desired reference trajectory. The RUV responds well and depth and heading are decoupled.

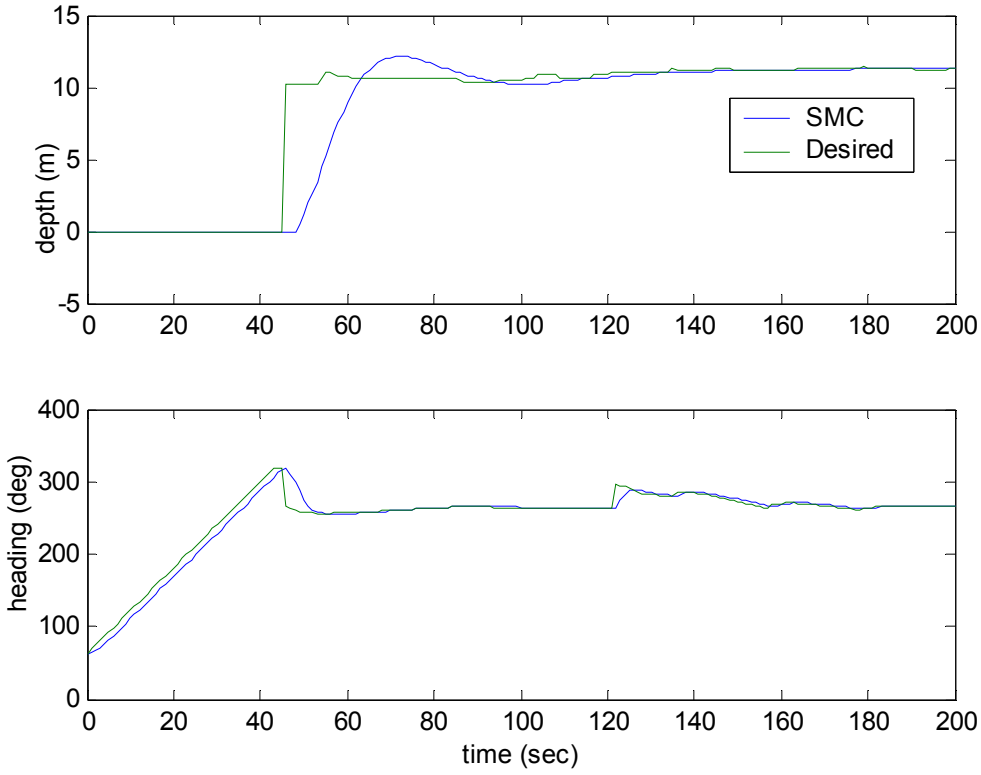

Figure 4.2. Regulation performance of the regular form sliding mode control law.

Controller Verification: Tracking

Figure 4.3 shows the tracking performance of the regular form sliding mode control law. The system is initialized at a depth of 0 meters and a heading of 180°. The RUV tracks well in both heading and depth using the regular form sliding mode controller. Note that there is a lag in both state variables. By letting the sliding surface be a function of the integral of the error, this tracking steady state error would be eliminated. Refer to [Slotine and Li] for details.

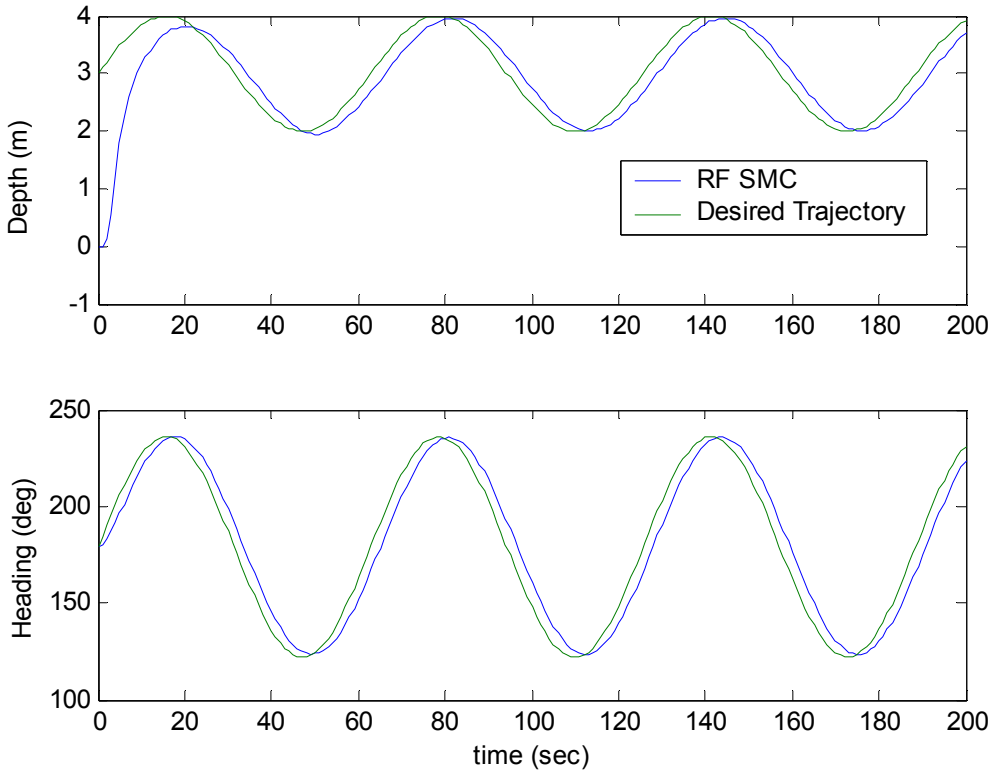

Figure 4.3. Tracking performance of the regular form sliding mode control law.

Controller Verification: Robustness

Parameter uncertainty is a paramount concern in developing tracking and regulation control laws for underwater robotic vehicles because hydrodynamic coefficients are difficult to obtain and may only be approximately known. The force and moment coefficients, as was shown in Chapter 3, are approximated semi-empirically or estimated in tow tanks [Prestero]. In order to simulate the effects of parameter

uncertainty on control performance, the following substitutions is introduced into the depth subsystem model

$$\begin{aligned} M'_{uq} &= -10 * M_{uq} \\ N'_{r|r|} &= 0.5 \cdot N_{r|r|} \end{aligned} \quad . \tag{4.51}$$

Note that the resulting modeling uncertainty is input to both matched and unmatched depth subsystem channels (Equations 4.47). Figure 4.4 shows the performance of the regular form sliding mode control law compared to the heading PD and depth P/PID control law developed for the model test of Chapter 3. Clearly the system utilizing the controller based on the linearized system performs poorly, with sustained oscillations occurring in the depth variable, i.e., the RUV appears to be marginally stable. The performance of the linear controllers when no model uncertainty exists is shown in Figure 3.10.

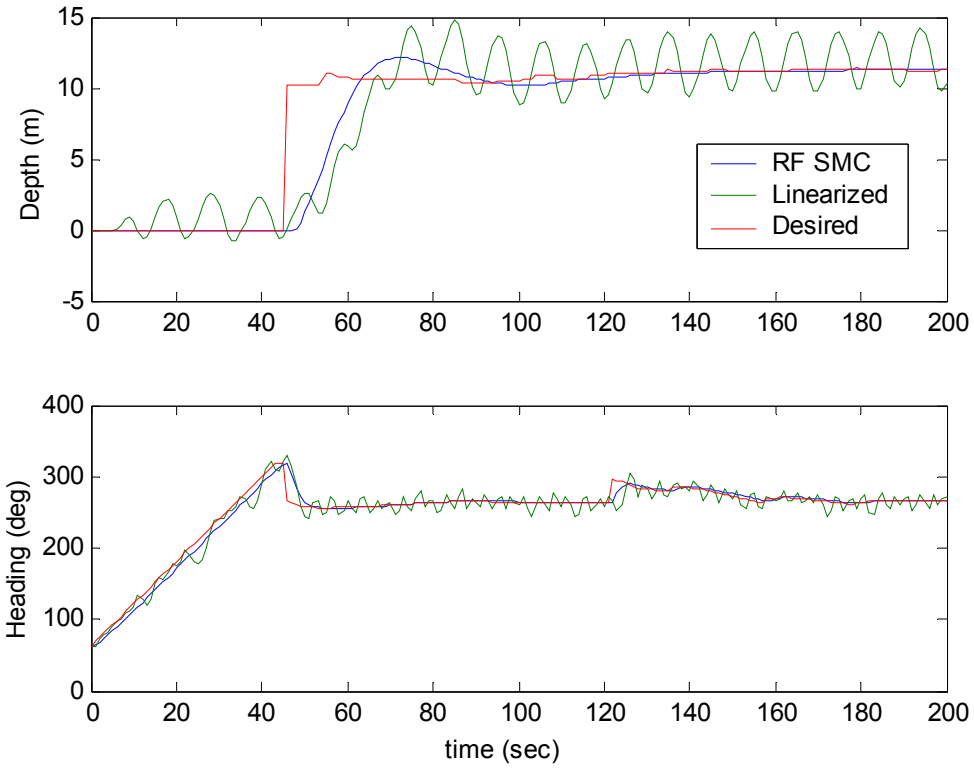

Figure 4.4. Robustness performance of the regular form sliding mode control law compared to the PID-type control law of Chapter 3 using the model uncertainty of Equation 4.51. The performance of the linear controllers when no model uncertainty exists is shown in Figure 3.10.

Chapter 5: Experimental RUV

INTRODUCTION

This chapter describes the mechanical, electronic and software features of an experimental robotic underwater vehicle and gives the results of the implementation and comparison of both conventional PID controllers and the regular form sliding mode controller. The purpose of the experimental RUV is to show that the more complicated formulas of the regular form sliding mode controller (Equations 4.36 and 4.50) can still be implemented real-time in an embedded system and that the controller's performance with regard to modeling uncertainty justifies the added complexity. The Appendix shows the RUV electronic schematics and lists the embedded code. Table 5.1 gives a selected component listing of the main sensors, motor drivers, motors, and integrated circuits.

VEHICLE DESCRIPTION

Mechanical

Figure 5.1 shows an image of the experimental RUV. A 32"x5" diameter acrylic tube comprises the vehicle's dry hull. The housing contains the electronics board and ballast weights. The wet hull shown in Figure 5.2 contains the pressure sensor, water-proof fin servos, and the thruster-propeller assembly. The end caps which seal the water-proof electronics housing are machined for double o-rings. Ballast weights are at the

electronics tray's fore and aft section. Twelve NiMh rechargeable batteries are located beneath the tray. The RUV cg is at the center of water proof section.

The fin actuator motors are simple hobby servos that have been modified to be water proof. The modifications include changing the shaft o-ring thickness. When the motor is reassembled the housing presses against the o-ring more tightly affecting a better face seal. Also, the seams of the servo housing assembly were sealed with a flexible urethane. The motors appear to be water proof to approximately 15-20 feet of water. This depth is adequate to demonstrate the control routines derived in previous chapters.

The fins are modified model airplane wings. They are constructed of lightweight foam and therefore provide buoyancy to the RUV in addition to control actuation.

The thruster is a 12V, 2.5A bilge pump motor. The bilge housing and impeller have been removed. A shaft was then machined and installed onto the existing motor shaft in order to extend the propeller 3 inches into the water beyond the motor housing (see Figure 5.3). This type of motor is useful for an experimental RUV since the motors are inexpensive and waterproof to around 40 feet of water.

Table 5.1 RUV Primary Components Parts Listing

Part Description	Part Number	Supplier
Pitch-Roll Sensor	CXTA01	Crossbow
Angle-rate sensor	ADXRS150EB	Analog Devices
Depth sensor	4040PC100G5D	Honeywell
Compass	Vector V2X	Precision Navigation, Inc
Fin servo	HS-755HB	HiTEC
Thruster	360GPH	Rule
3A 55V H Bridge	LMD18299T	Digikey
5V DC converter	UNS-5/3-D12	Datel
ADAPT11	AD11FM	Technological Arts
DOS Stamp	DOS Stamp	Bagotronix

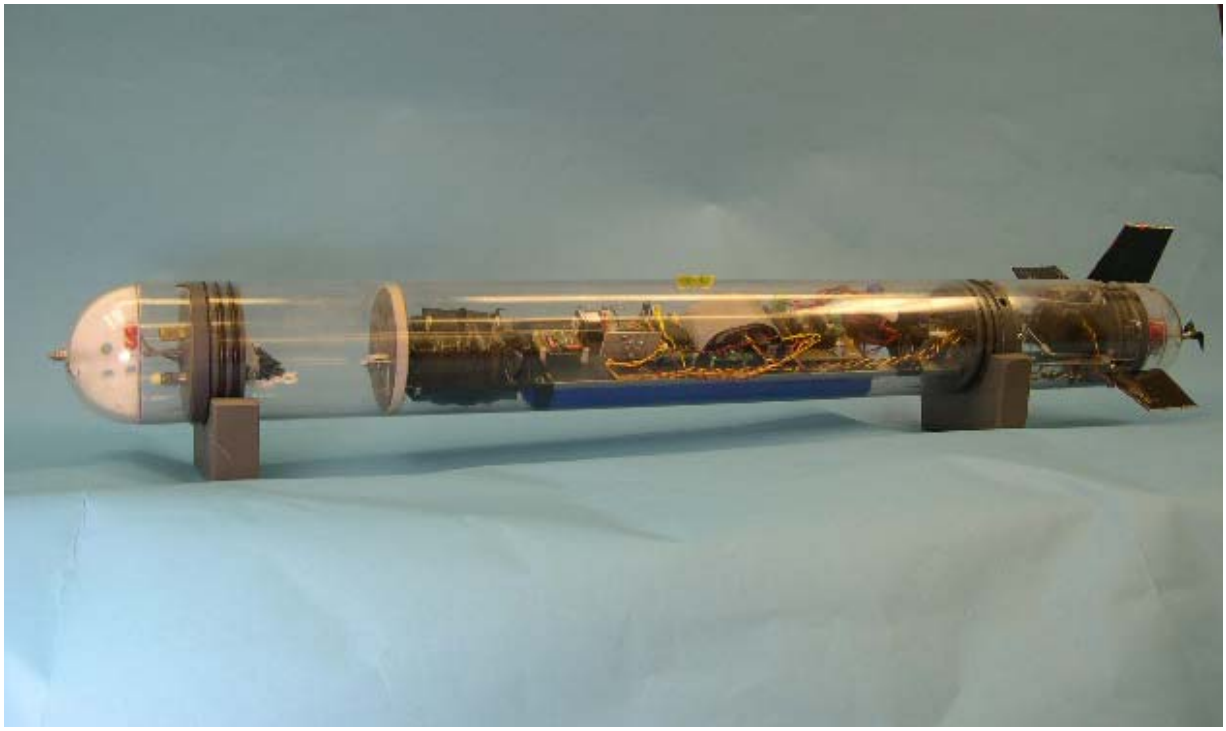

Figure 5.1. Image of the experimental RUV. The RUV was developed to test the hypothesis that the regular form sliding mode control law could be implemented in an embedded environment and that the controller would outperform the conventional PID approach in terms of robustness to model uncertainty.

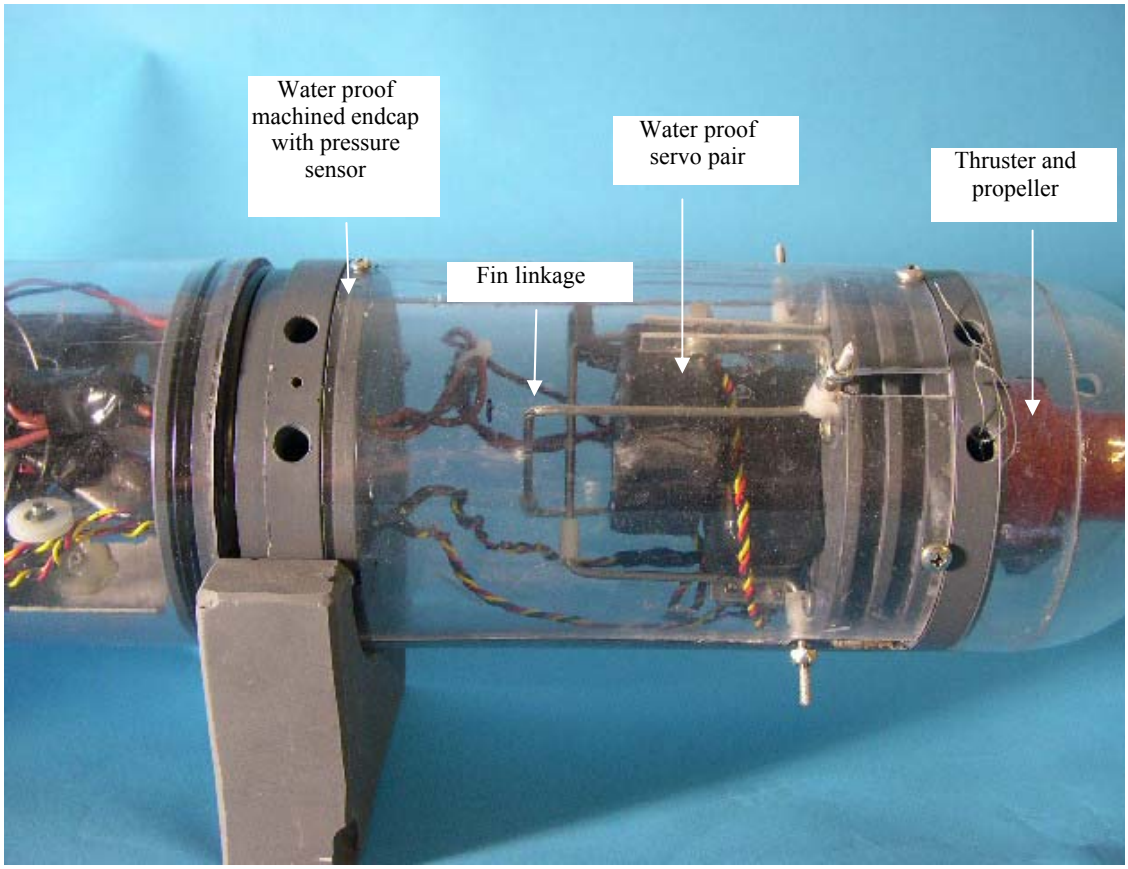

Figure 5.2 The RUV servo and thruster assembly. This wetted portion of the RUV hull, shown with fins removed, houses the water-proof fin servos, fin linkage and the single thruster motor.

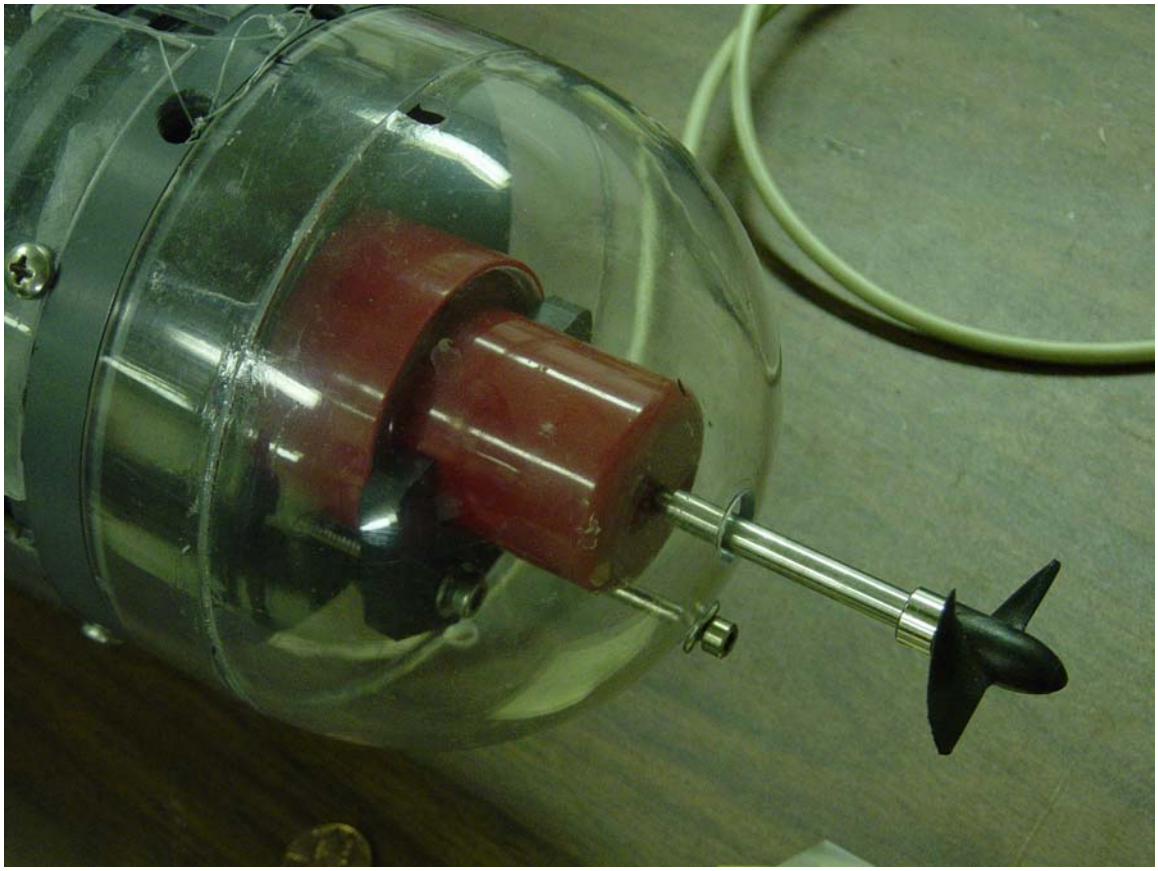

Figure 5.3 Thruster, shaft and propeller assembly shown with fins removed. The thruster utilizes an off-the-shelf bilge pump motor and hobby-grade propeller.

Electronics

Figure 5.4 Shows a close-up of the electronics package. The electronic hardware consists of two bus-connected microcontrollers, a custom built circuit card, and various sensors.

The microcontroller pair consists of a DOS Stamp and the Adapt11, a Motorola 6811-based controller. The stamp features a 40 MHz CPU (512K SDRAM, 2 DMA channels, power save modes, 3 timer/counters, 2 UARTs, general purpose I/O, 80186 instruction set) running an embedded version of DOS. Thus, it has useful directory and file utilities. The stamp executes the mission control code as well as routines for reading the yaw-rate, pitch-rate and pitch sensors. The Adapt11 runs the code for generating the thruster and servo PWM signals and interfacing the compass through its SPI port.

The custom circuit card has a 5V dc-dc converter, charger circuitry, power management circuitry, and a pulse-width modulation amplifier for generating thruster power. The dc converter supplies regulated power to the microcontrollers, fin servos, and various sensors. The Appendix contains the circuit schematics. Battery charging and RS232 communications with an external PC are facilitated by closing the TX, RX and charger reed switches with a magnet placed near the RUV hull at the corresponding reed switch location on the circuit card (Figure 5.4). The external PC can upload new code and mission plan text files, as well as download log files for analysis.

The sensors include yaw rate, pitch-rate, pitch/roll angle, fluxgate compass, and pressure. All of these sensors are 5V, pre-amplified devices.

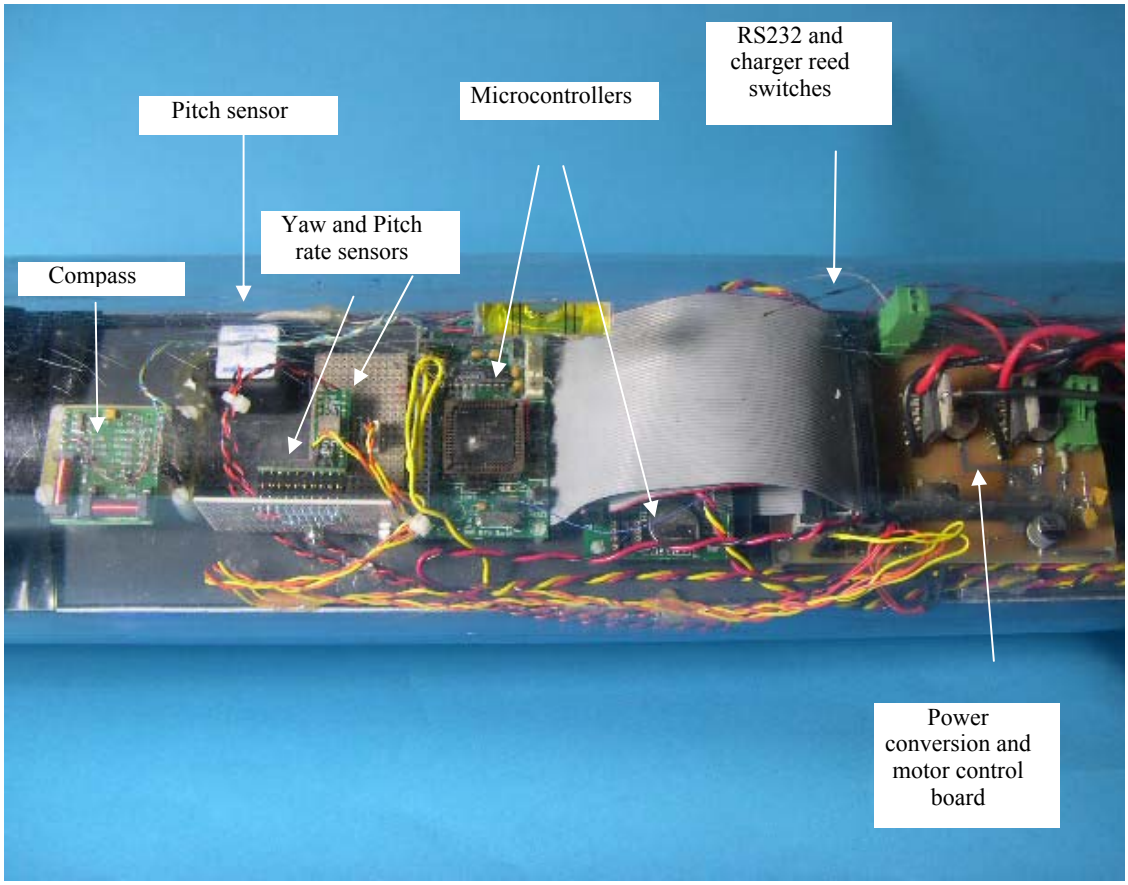

Figure 5.4. The RUV electronics package consists of a two bus-connected microcontrollers, sensors, motor controllers, and power conversion/management circuitry.

Software

The Appendix lists the C and assembly code used with the RUV. The assembly routine is for the 6811 microcontroller. It is an interrupt-driven routine that generates pulse-width modulated signals, on/off and direction commands for the thruster driver and the two fin servos. Additionally the routine drives the analog-to-digital converters, the digital input/output lines and interfaces the compass through the serial peripheral interface port. Serial drivers allow the microcontroller to communicate with the DOS Stamp via the shared RS232 lines.

The DOS Stamp helper routines for standard serial communication, memory access, interrupt handling, analog and digital port conversion, and real-time clock operation consist of DOSSTAMP.C, DSADC.C, DSRTC.C and TDSCOMM.C. These driver routines were written by the DOS Stamp developers and are included with the DOS Stamp starter kit [Bagotronix]. The RUV application-specific DOS Stamp routines contained in ADAPT11.C communicate with the ADAPT11 microcontroller to run the thruster and servos. Additional routines read the depth, pitch, roll, pitch- and yaw-rate, and compass sensors connected to ADC ports on the DOS Stamp. Finally, ADAP11.C contains code for both PID and sliding mode control autopilot routines. The routines in AUVMODEL.C calculate the model state for the sliding mode controller as well as the regular form coordinate transformation and its Jacobian. The numerical routines in NLIB.C are found in [Shilling and Harris]. The code has vector and matrix data types,

functions for dot products, matrix multiplication and inversion which facilitate the real-time sliding mode computations.

COMPARISON: PID VS. SLIDING MODE EMBEDDED CONTROL

The purpose of the experimental RUV is to show that the more complicated formulas of the regular form sliding mode controller can still be implemented real-time on an embedded system and that the controller's performance with regard to modeling uncertainty justifies the added complexity. Simply implementing and running the sliding mode depth and heading autopilot routines of Chapter 4 on the experimental RUV proves the first assertion. Clearly the 1Hz control loop rate indicates that the system could benefit from a faster CPU with a math co-processor.

In order to test the second assertion, two models are utilized. The models differ by changing the cross cylinder form drag coefficient estimate used in calculating Equations 3.30 and 3.31 by a factor of 6. The resulting force and moment coefficients for each model are listing in Tables 5.3 – 5.6.

Experimental Setup

The data were collected in a residential pool on November 7th, 2006. The pool dimensions allowed for approximately 13.3 meters of travel. The nominal vehicle speed was around 0.25 m/s (0.5 knot) yielding approximately 54 second runs. The depth autopilot routine was not engaged due to excessive vehicle pitch resulting from a noisy depth sensor. The vehicle pitch reduces the compass accuracy by 3° for every 1° of tilt

since the Vector V2X is not gimbaled. For each run the RUV was placed at rest approximately 0.3 meters deep at zero pitch. The test conditions were characterized by the following features:

- No wind, zero current
- No wave perturbation
- Modeling uncertainty
- Sensor noise
- Control loop time delays: 1.47 Hz for PID, 1 Hz for Sliding Mode

Note that the control loop time delays are increased because of the RUV state data logging operations. With logging disabled the simple PID formulas allow the update rate to be approximately 5 Hz. Several features of the RUV setup included:

- No monofilament retrieval line attached to RUV
- Only heading auto-pilot enabled
- 0.25 knot nominal speed (Open loop speed control)
- Both control methods utilize the model coefficients (Model A, Tables 5.3 and 5.4 and Model B, Tables 5.5 and 5.6) calculated with the code listed in the Appendix, *Calc_5in_RUV_params.m*.

The Model A proportional and derivative controller gains for the heading PD autopilot are, respectively, $K_{p,heading} = -54.50$ and $K_{d,heading} = -11.05$. The Model B proportional and derivative controller gains are $K_{p,heading} = -54.50$ and $K_{d,heading} = -15.70$, respectively.

The regular form sliding mode control parameters are as follows. The sliding mode heading controller utilizes a constant switching term gain of 10. Three different boundary layer values were utilized in Equation 4.15 for the performance term: δ = 1, 10, and 20. The heading subsystem sliding mode surface coefficient vector is $\overline{s}_h = \begin{bmatrix} 1.0 & 1.0 & 1.0 \end{bmatrix}^T$.

Table 5.2 lists the pertinent RUV parameters for calculating the force and moment coefficients using the formulas developed in Chapter 3. Models A and B differ in the c_{dc} parameter, noted in parenthesis. The resulting moment and force coefficient values for both models are listed in Tables 5.3 and 5.4, respectively, with different model parameters noted in parenthesis.

Table 5.2. Experimental RUV Parameters

Parameter	Value	Units	Description
ρ	1010	g/m^3	Fluid density
g	9.81	m/s^2	Gravitational acceleration
x_n	0.597	m	Distance form origin to vehicle nose
x_t	-0.648	m	Distance form origin to vehicle tail
x_{fin1}	-0.559	m	Distance form origin to fin trailing edge
x_{fin2}	-0.484	m	Distance form origin to fin leading edge
d	0.127	m	Max body diameter
l	1.24	m	Vehicle body length
S_w	0.497	m^2	Approximate wetted surface area (πdl)
A_p	0.158	m^2	Approximate hull planform area (ld)
A_f	0.013	m^2	Approximate frontal area ($\pi d^2/4$)
c_{ds}	0.004	-	Skin drag coefficient
c_{dF}	0.3	-	Axial drag coefficient referenced to A_f
c_{da}	0.0125	-	Total axial drag coefficient ref. to S_w
c_{dc}	0.1745 (A) 0.0324 (B)	-	Total crossflow coefficient ref. to S_w
l_{cp}	0. 2121	m	Distance to center of pressure
$\overline{r}_f$	0.131	m	Distance from vehicle axial center to fin center
R_{fin}	0.127	m	Distance from vehicle axial center to fin tip
w_{fin}	0.0890	m	Width of fin base
t_{fin}	0.0635	m	Width of fin tip
h_{fin}	0.0635	m	Fin height
A_{fin}	0.0048	m^2	Rudder/stern fin area
A_r, A_s	0.0048	m^2	Rudder/stern fin planform area
c_{df}	1.56	-	Fin form drag coefficient
c_r, c_s	1.79	-	Rudder/stern fin lift coefficient
l_{fin}	0.5335	m	Fin moment arm
W	$1.25x10^2$	N	Weight
B	$1.25x10^2$	N	Buoyancy
x_b	0	m	x-component center of buoyancy
y_b	0	m	y-component " " "
z_b	0	m	z-component " " "
x_g	0.001	m	x-component center of gravity
y_g	-0.001	m	y-component " " "
z_g	0.0508	m	z-component " " "
I_{xx}	0.0196	kg m^2	Moment of inertia about x axis
I_{yy}	7.26	kg m^2	Moment of inertia about y axis
I_{zz}	7.26	kg m^2	Moment of inertia about z axis
R	4.32E-2	m	Propeller radius
u_0	1.0	m/s	Design speed

Table 5.3. Experimental RUV Moment Coefficients

Parameter	Value	Units	Description
$K_{p\|p\|}$	-2.60	kg m^2/rad^2	Rolling drag
$K_{\dot{p}}$	-0.063	kg m^2/rad^2	Added mass
$M_{w\|w\|}$	2.95 (A), 3.86 (B)	kg	Cross flow drag
$M_{q\|q\|}$	-34.48 (A), -7.35 (B)	kg m^2/rad^2	Cross flow drag
M_{uw}	11.86	kg	Body and fin lift + Munk Moment
$M_{\dot{w}}$	2.03	kg m	Added mass
$M_{\dot{q}}$	-2.92	kg m^2/rad	Added mass
M_{uq}	-4.52	kg m/rad	Added mass cross term + fin lift
$M_{uu\delta_s}$	-4.67	kg/rad	Fin lift moment
$N_{v\|v\|}$	-2.95 (A), -3.86 (B)	kg	Cross flow drag
$N_{r\|r\|}$	-27.81 (A), -6.11 (B)	kg m^2/rad	Cross flow drag
N_{uv}	-11.86	kg	Body and fin lift + Munk Moment
$N_{\dot{v}}$	-2.03	kg m	Added mass
$N_{\dot{r}}$	-2.92	kg m^2/rad	Added mass
$N_{uu\delta_r}$	-4.67	kg rad	Fin lift moment

Table 5.4. Experimental RUV Force Coefficients

Parameter	Value	Units	Description
$X_{u\|u\|}$	-3.14	kg/m	Axial drag
$X_{\dot{u}}$	-0.212	kg	Added mass
$Y_{v\|v\|}$	-445.32 (A), -88.93 (B)	kg/m	Cross-flow drag
$Y_{r\|r\|}$	2.86 (A), 2.30 (B)	kg m/rad^2	Cross-flow drag
Y_{uv}	-22.41	kg/m	Body and fin lift
$Y_{\dot{v}}$	-19.64	kg	Added mass
$Y_{\dot{r}}$	-2.03	kg m/rad	Added mass
Y_{ur}	4.46	kg/rad	Added mass cross term + fin lift
$Y_{uu\delta_r}$	8.75	kg/m/rad	Fin lift force
$Z_{w\|w\|}$	-445.32 (A), -88.93 (B)	kg/m	Cross flow drag
$Z_{q\|q\|}$	-2.86 (A), -2.30 (B)	kg m/rad^2	Cross flow drag
Z_{uw}	-22.41	kg/m	Body and fin lift
$Z_{\dot{w}}$	-19.64	kg	Added mass
$Z_{\dot{q}}$	2.03	kg m/rad	Added mass
Z_{uq}	-4.46	kg/rad	Added mass cross term + fin lift
$Z_{uu\delta_s}$	-8.75	kg/m/rad	Fin lift force

Results

The results of eight experimental RUV runs are presented in this section. The pool dimensions limited the allowable heading range to around 36°. Thus the initial heading for each run was around 210° and the heading set point was 174°. Figure 5.5 shows the results using the force and moment coefficients from Model A for the three different values of the boundary layer parameter discussed in Chapter 4. The best steady-state heading rms error of 3.56° is achieved with $\delta = 10$. In the case of Model B, the sliding mode controller achieves an rms of 2°, shown in Figure 5.6. This is approximately the V2X compass measurement rms. The PD rms for Model A is 8° and is 6° for Model B (Figure 5.7). Clearly the sliding mode technique outperforms the conventional PD approach for the given two models.

Naturally one can tune the PD/PID heading and depth autopilot gains experimentally using, for example, the Ziegler-Nichols rules with experimental step responses [Ogata]. However, one can argue that a model-based approach allows one to easily retune the controller each time a new payload or hull configuration is required. If this convenience is sought, the data presented indicates that a sliding mode controller could better handle model uncertainties since for two different models, the sliding mode autopilot outperforms the PD heading autopilot. With the sliding mode technique, there remains the selection of the boundary layer parameter, δ. However, this value can be estimated in the model simulation by selecting a value which prevents the fin actuators from saturating for all nonzero values of σ in Equation 4.15.

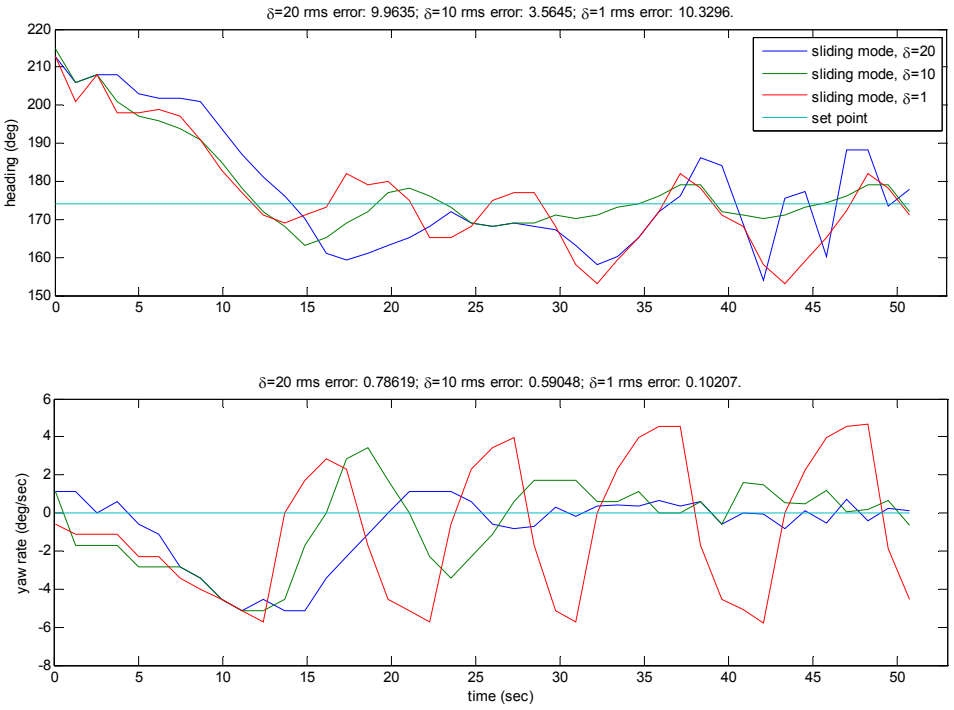

Figure 5.5. Experimental RUV closed loop heading response using the regular form sliding mode control with Model A parameters. The response in heading and yaw rate for three different values of the boundary layer control parameter are shown (δ in Equation 4.15).

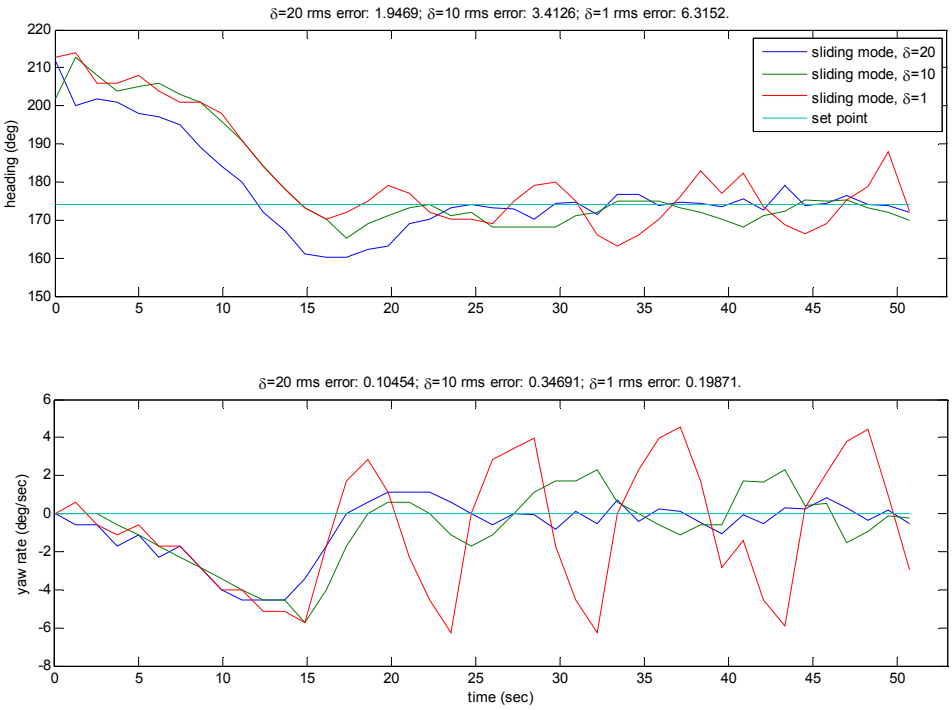

Figure 5.6. Experimental RUV closed loop heading response using the Chapter 4 sliding mode control with Model B parameters. The response in heading and yaw rate for three different values of the boundary layer control parameter are shown (δ in Equation 4.15).

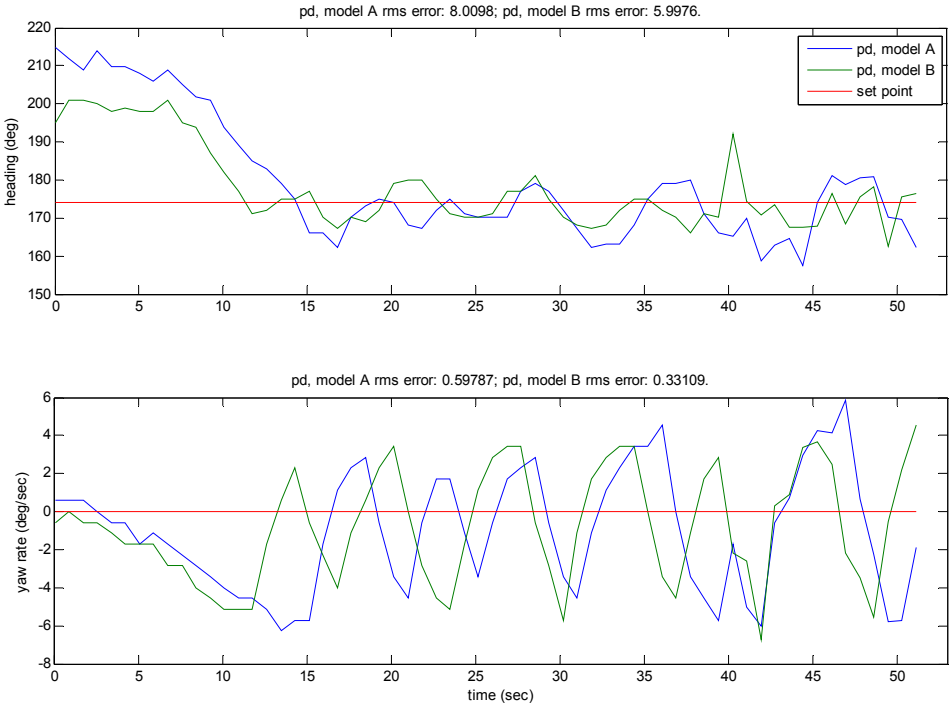

Figure 5.7. Experimental RUV closed loop heading response using the Chapter 3 proportional-derivative heading controller. Responses for both models are shown. The best error response is 5.99 steady-state heading rms for Model B.

Chapter 6: Optimal Gain Sliding Mode Control

OPTIMAL SLIDING MODE CONTROL CONCEPT

Sliding mode control is a robust control technique offering guaranteed stability according to Theorem 4.3 which specifies the conditions the performance term gain, η, (Equation 4.13) is required to satisfy. Typically the sliding mode control design specifies a fixed gain for the performance control term, u_p, in Equation 4.11 based on a conservative estimate of the maximum uncertainty associated with the dynamic model estimate $\hat{f}$ used in the control law formulation. However, an "optimal" gain sliding mode control technique results if the gain is adjusted dynamically as a function of the state-dependent model uncertainty. The resulting controller reduces the gain, decreasing the required control power and control chatter [Slotine and Li] compared to a constant gain approach. Figure 6.1 illustrates the difference between sliding mode control gain selection based on fixed conservative estimate of model uncertainty and the optimal time varying approach.

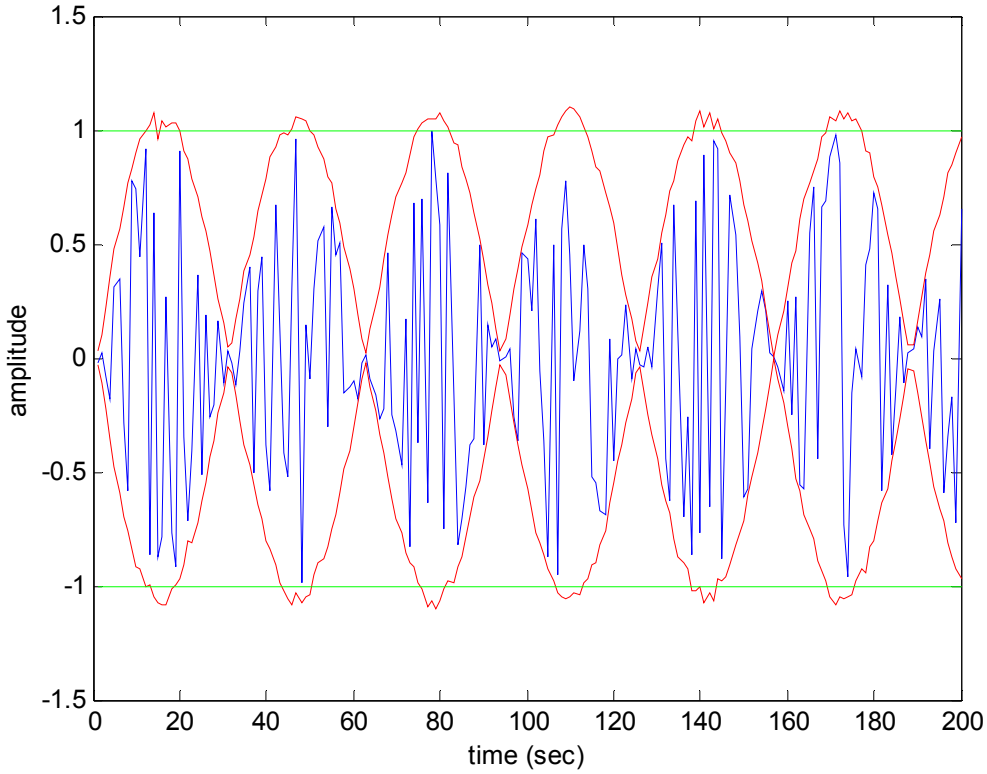

Figure 6.1. Illustration of the nonstationary nature of uncertainty bounds. A noisy system state (blue) has uncertainty bounds (red) that vary in time (not necessarily symmetric). A typical sliding mode control design bases the selection of the performance term gain on a conservative constant estimate of the maximum dynamic uncertainty (green).

Clearly, for a given set of sliding surface coefficients, the sliding mode control performance term gain conditioned on the actual uncertainty bounds (red time series Figure 6.1) will minimize the control energy performance index while satisfying the stability criterion of Theorem 4.3. Hence it is the least active (optimal) controller possible that guarantees performance.

OBTAINING THE OPTIMAL UNCERTAINTY ESTIMATE

For control purposes the dynamic uncertainty can result from parametric uncertainties (inaccuracies in the terms included in the model), unstructured uncertainties (inaccuracies in the model order) [Slotine and Li], and external unmodelled disturbances. In some situations an estimate of the model dynamic uncertainty bounds can be obtained, either from first principles or through system identification techniques. For example, [Buckner, Fernandez, Masada] and [Fernandez and Buckner] present an offline regression method for estimating the confidence interval related to modeling uncertainties for use in the optimal sliding mode control gain using radial basis neural networks. The approach utilizes a specially designed RBNN with a biased learning error that regresses the $\pm 2\sigma$ values of the modeling uncertainty instead of the customary average value of the noisy system dynamics. However, this technique assumes that the full state is accurately *measurable* during online implementation.

Problems occur with methods that utilize analytical models or offline identification techniques to produce a state-dependent bounds estimate when only noisy state measurements are available at run time because such techniques cannot optimally fuse dynamic model predictions with noisy measurements. A stochastic state space formulation in combination with a requirement to update state information on receipt of new measurements is ideally suited to the sequential Bayesian estimation approach. The basic method involves propagating in time the probability density function of the state, the so called hyperstate probability density function (PDF), given all available

information including the set of received measurements [Lewis 1986]. The formulation of the problem in this fashion leads to a statement of the Sequential Bayes Estimation Problem (SBEP). The SBEP is characterized in terms of a process and measurement model and the hyperstate pdf given by

$$\begin{aligned} x_k &= f(x_{x-1}, u_{k-1}, w_{k-1}) \\ z_k &= h(x_k, v_{k-1}) \\ p(x_k \mid z_k) \end{aligned} \quad , \tag{6.1}$$

respectively, where w and v are the process and measurement noise processes, respectively. The prediction stage of the SBEP involves propagating the hyperstate pdf from time *k-1* to *k* by evaluating the integral

$$p(x_k \mid z_{k-1}) = \int p(x_k \mid x_{k-1}) p(x_{k-1} \mid z_{k-1}) dx_{k-1} \tag{6.2}$$

where the state transition pdf is given by

$$\begin{aligned} p\left(x_k \middle| x_{k-1}\right) &= \int p(x_k \mid x_{k-1}, w_{k-1}) p(w_{k-1}, x_{k-1}) dw_{k-1} \\ &= \int p(x_k \mid x_{k-1}, w_{k-1}) p(w_{k-1}) dw_{k-1} \\ &= \int \delta\left(x_k - f(x_{x-1}, u_{k-1}, w_{k-1})\right) p(w_{k-1}) dw_{k-1}. \end{aligned} \tag{6.3}$$

Next, a measurement is received and the hyperstate pdf is updated by using Bayes Rule

$$p(x_k \mid z_k) = \frac{p(x_k \mid z_{k-1}) p(z_k \mid x_k)}{\int p(x_k \mid z_{k-1}) p(z_k \mid x_k) dx_k}. \tag{6.4}$$

Similarly the measurement PDF is evaluated as

$$p(z_k \mid x_k) = \int \delta\left(z_k - g(x_x, v_k)\right) p(v_k) dv_{k-1}. \tag{6.5}$$

A solution to the SBEP minimizes the mean square estimation error defined as the expected value of the Euclidean norm square of the error $\left(x_k - \hat{x}_k\right)$

$$J_{MSE} = E\left[\left(x_k - \hat{x}_k\right)^T \left(x_k - \hat{x}_k\right)\right] \tag{6.6}$$

where x_k is the true state and $\hat{x}_k$ its estimate.

Relating the Covariance Estimate and Modeling Uncertainty

In the conventional approach to state estimation, a dynamic model is used to predict the system state and then the prediction is compared to an actual measurement to obtain the optimal state estimate at time *k*. However, in order to obtain a useful state and gain estimate for the optimally-robust control method, the propagation and measurement steps are reversed. That is, a measurement at time *k* is compared to the model prediction stored from time *k-1*. The state estimate and error covariance are updated with this information and the (optimal) state estimate is used for state feedback in the controller.

Next, the state estimate and error covariance are propagated to time *k+1* via the dynamic model equations. The propagated error covariance is given by

$$\begin{aligned}\hat{P}_f &= E\left[\left(x_{k+1}-\hat{x}_{k+1}\right)\left(x_{k+1}-\hat{x}_{k+1}\right)^T\right] \\ &= E\left[\left(f(x_k)-\hat{f}(\hat{x}_k)\right)\left(f(x_k)-\hat{f}(\hat{x}_k)\right)^T\right] \\ &\equiv E\left[\left(f_k-\hat{f}_k\right)\left(f_k-\hat{f}_k\right)^T\right].\end{aligned} \tag{6.7}$$

With the identity $trace(P) = E[x^T x]$ one obtains

$$trace(\hat{P}_f) = E\left[\left(f_k-\hat{f}_k\right)^T\left(f_k-\hat{f}_k\right)\right]. \tag{6.8}$$

Let $\|\cdot\|$ denote the L_2 norm whose definition for a real, N-component vector is

$$\|x\| = \sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{x^T x}\,. \tag{6.9}$$

Then

$$trace(\hat{P}_f) = E\left[\|\Delta f_k\|^2\right]. \tag{6.10}$$

Therefore, the trace of the covariance matrix is related to the average modeling uncertainty and can be used to dynamically adjust the gain in the performance

(switching) term of the optimally-robust nonlinear RUV control law. The switching term gain at time *k* satisfies

$$\eta_k = \|\bar{s}\| E\left[\|\Delta f_k\|\right] + \varepsilon, \quad \varepsilon > 0 \tag{6.11}$$

where $E\left[\|\Delta f_k\|\right]$ is the expected modeling uncertainty and ε is a term that guarantees performance (e.g. it bounds reaching time to the sliding surface). This replaces the corresponding deterministic quantity in Theorem 4.3 since Δf_k is now a random variable. Also, note that η_k is in the regular form transformation variables. Since

$$trace\left(\hat{P}_f\right) = E\left[\|\Delta f\|\right]^2 - \gamma \tag{6.12}$$

where $\gamma > 0$, one can simply multiply the left hand side by a positive scalar constant to obtain the bound

$$\alpha \cdot \sqrt{trace\left(\hat{P}_f\right)} \geq E\left[\|\Delta f\|\right]. \tag{6.13}$$

Substituting Equation 6.13 from above gives

$$\eta_k' = \alpha \|\bar{s}\| \sqrt{trace\left(\hat{P}_f\right)} + \varepsilon', \quad \varepsilon' > 0 \tag{6.14}$$

where the constant of proportionality, α, can be determined from simulation.

Solution to Sequential Bayes Estimation Problem

In the general case there does not exist an analytic solution to the SEBP as the integrals in Equations 6.2-6.5 are intractable. However, in the situation where the process and measurement models are linear with zero mean Gaussian system/plant and measurement noise, the estimate that minimizes J_{MSE} can be obtained by the Kalman Filter [Lewis 1986]. When the system or measurement model is nonlinear and/or the noise processes are non-Gaussian, then the Kalman Filter is suboptimal. In situations where nonlinearities exist, the KF is often replaced with the Extended Kalman Filter (EKF). Table 6.1 lists the equations of the Extended Kalman Filter operating in a single step ahead prediction mode. This predictor can be used for the state and gain estimation within the nonlinear sliding mode control law by using the state estimate, $\hat{x}_k$, at current time for state feedback and the uncertainty variance, $\hat{P}_f$, for the optimal control gain in Equation 4.13.

Table 6.1. The Extended Kalman State and Gain Observer

Initialization Step:	
$\dot{x}(t) = f(x(t)) + g(x(t))U(t) + w(t)$ $y(t) = h(x) + v(t)$	Process and Measurement model
$\overline{w}_k = 0,\ E[w_k w_j^T] = Q\delta_{kj}$ $\overline{v}_k = 0,\ E[v_k v_j^T] = R\delta_{kj}$ $E[v_k w_j^T] = 0, \quad E[x_k w_j^T] = 0, \quad E[x_k v_j^T] = 0$	Process and Measurement noise model assumptions
$\hat{x}_0, \quad \hat{P}_0$	Initial state and model uncertainty estimate
Measurement update at time k:	
$r_k = y_k - h(\hat{x}_k)$	Residual
$H_k = \left.\frac{\partial h}{\partial x}\right\|_{\hat{x}_k}$	Kalman gain
$K_k = \hat{P}_k H_k \left(H_k \hat{P}_k H_k^T + R\right)^{-1}$ $\hat{x}_k = \hat{x}_k + K_k r_k$ $\hat{P}_k = (I - K_k H_k)\hat{P}_k$	Updated state estimate: $\hat{x}_k$ **sent to controller for state feedback**
Prediction at time k:	
$\dot{x}(t) = f(x) + gu, \quad IC: x(0) = \hat{x}_k$ $A_k = \left.\frac{\partial f}{\partial x}\right\|_{\hat{x}_k}$ $\dot{\Phi}(t, t_k) = A_k \Phi(t, t_k), \quad IC: \Phi(0,0) = I$	Optimal one step ahead model uncertainty prediction step: $\hat{P}_f$ **sent to controller for gain estimate**
$\hat{P}_{k+1} = \Phi(t_{k+1}, t_k)\hat{P}_k \Phi^T(t_{k+1}, t_k) + Q$ $\hat{P}_f = \hat{P}_{k+1}$	

In the case of nonlinear process and/or measurement model and non-Gaussian statistics, an improved approach to achieve state and gain estimates for use in the sliding mode controller is to use a single-step ahead predictor Particle Filter. The basic Particle Filter (PF) is a recursive algorithm for estimating the entire system probability density function (PDF) by representing the required PDF as a set of random samples or particles [Gordon, et. al.]. Each particle represents a possible state of the system and is propagated individually through the system state equations. Each particle is then compared to the actual state measurement via the measurement likelihood function and re-weighted accordingly. The effect is similar to the outcome of the Kalman Filter (KF) algorithm. However, instead of propagating only the system mean and variance as with the KF, an approximation to the entire PDF is propagated, allowing one to compute not only the mean and variance, but any function of the PDF. The advantage of the PF over the KF is that one may treat nonlinear dynamic and measurement models without linearization approximations. Also PFs can handle non-Gaussian process and measurement noise. The disadvantage is that in some cases hundreds of particles are required to accurately approximate the PDF which can slow processing time.

In order to achieve state and gain estimates the basic Particle Filter algorithm presented in [Gordon, et. al.] is modified to perform as a single-step ahead predictor. This is achieved by first implementing the measurement cycle first followed by the prediction cycle. Table 6.2 lists the algorithm where k is the time index.

Measurement Update at time k: First compute the likelihood of each particle's residual, i.e., the difference between the predicted particle observables and the measured

value of the system's state. This requires that the likelihood function, $p\left(z_k \middle| x_k\right)$, be of a known functional form. Next compute a new set of normalized weights $\tilde{\omega}_k^{(i)} = p\left(z_k^{(i)} \middle| \tilde{x}_k^{(i)}\right) \Big/ \sum_{i=1}^{N} p\left(z_k^{(i)} \middle| \tilde{x}_k^{(i)}\right)$, yielding a new random measure $\left\{\tilde{x}_k^{(i)}, \tilde{\omega}_k^{(i)}\right\}_{i=1}^{N}$. Now resample N times from this discrete distribution to generate samples $\left\{x_k^{(i)}\right\}_{i=1}^{N}$ such that for any *j*, $\Pr\left\{x_k^{(j)} = \tilde{x}_k^{(i)}\right\} = \tilde{\omega}_k^{(i)}$. The resulting random measure, $\left\{x_k^{(i)}, N^{-1}\right\}_{i=1}^{N}$, approximates the pdf $p\left(x_k \middle| z_k\right)$ as desired. The mean of the resulting approximate distribution, $\overline{X}_{PF,k}$, is sent to the controller for state feedback (c.f. Table 6.2).

Prediction Step at time k:: Each particle is propagated through time via the system process equation $x_{k+1}^{(i)} = f\left(x_k^{(i)}, u_k, w_k^{(i)}\right)$ where $w_k^{(i)}$ is sampled from the process noise distribution. The continuous process equations in Table 6.2 can be discretized or integrated numerically with small time steps to simulate discretization. The propagated set of particles yields the random measure $\left\{\tilde{x}_{k+1}^{(i)}, N^{-1}\right\}_{i=1}^{N}$ characterizing the pdf $p\left(x_{k+1} \middle| z_k\right)$. The covariance of this approximate distribution, $\hat{P}_f$, is computed and utilized for the controller gain as a measure of the functional uncertainty bound.

Table 6.2. The Particle Filter State and Gain Observer

Initialization:

$$\dot{x}(t) = f(x(t)) + g(x(t))U(t) + w(t)$$
$$y(t) = h(x) + v(t)$$

Process and Measurement model

$$\overline{w}_k = 0,\ E[w_k w_j^T] = Q\delta_{kj}$$
$$\overline{v}_k = 0,\ E[v_k v_j^T] = R\delta_{kj}$$
$$E[v_k w_j^T] = 0, \quad E[x_k w_j^T] = 0, \quad E[x_k v_j^T] = 0$$

Process and Measurement noise model assumptions

**The Particle Filter PDF X_{PF} is initially approximated by N random samples normally distributed about the first noisy state measurement.

Initial state and model uncertainty estimate

Measurement Update at time k:

For each particle:

Compute the residual

Compute the likelihood

Normalize the resulting distribution

Resample the distribution

Each of these procedures is discussed in [Gordon, et. al.]. Contact the dissertation author for the Matlab code.

Compute the mean:

$$\hat{x}_k = \overline{X}_{PF,k}$$

Updated state estimate: $\hat{x}_k$ **sent to controller for state feedback**

Prediction Step at time k:

For each particle:

Integrate the system equations over the time step

The set of final states approximates the PDF

Compute the variance:

$$\hat{P}_f = Cov(X_{PF,k+1})$$

Optimal one step ahead model uncertainty prediction step: $\hat{P}_f$ **sent to controller for gain estimate**

Figure 6.2 illustrates graphically the PF state and gain observer prediction and update cycle for *N*=8 particles. The large circles just before the resample phase represent the particles after re-weighting. The re-weighting is proportional to the likelihood that a given particle resulted in the observed measurement quantified in the expression for $p(z_k \mid x_k)$. The more likely a particle given the observation, the more weight it is given, and subsequently the more particles it spawns during the resampling phase ("Resample" box, Figure 6.2).

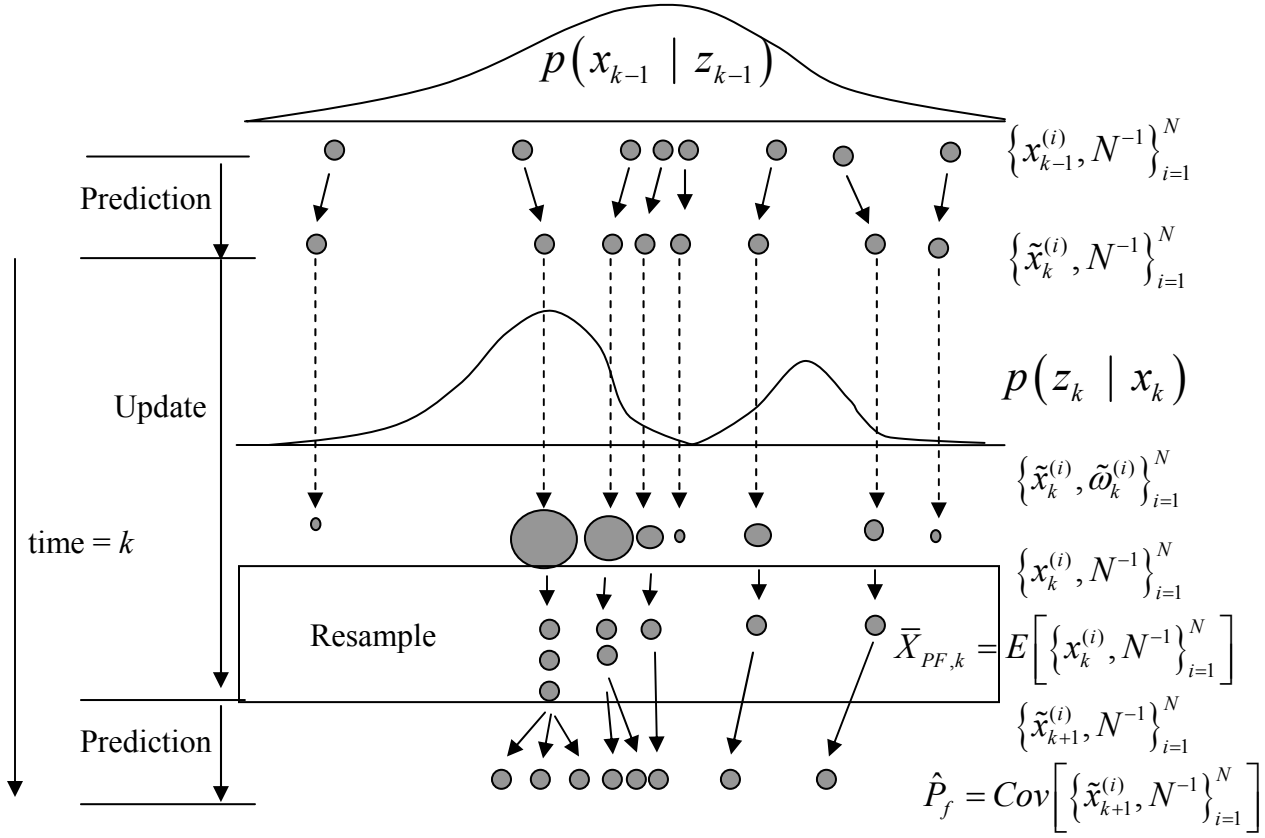

Figure 6.2. The Particle Filter state and gain observer timing diagram. The prediction, update, resample and prediction cycle for N=8 particles is shown. The variable k represents the discrete time index. The notation $E[\ ]$ and $Cov[\ ]$ represents the expectation and covariance operation, respectively. The remaining notation on the right hand side of the diagram is explained in the text.

Example: Comparison between EKF and PF State and Uncertainty Bounds Observers

This section presents the results of the a simulation illustrating the nonlinear model uncertainty bounds predictive ability of the Particle Filter state and bounds estimation algorithm listed in Table 6.2. A comparison is made with the Extended Kalman Filter algorithm listed in Table 6.1. As will be shown, the PF presents a distinct advantage over the EKF since it is able to track highly nonlinear plant trajectories and provide a more accurate uncertainty bound. The system under investigation is the following open-loop, non-autonomous, nonlinear, discrete time plant driven by zero mean, white noise process and measurement inputs, w_k and v_k, respectively [Gordon, et. al.]:

$$\begin{aligned} x_{k+1} &= 0.5x_k + 25x_k \Big/ \left(1 + x_k^2\right) + 8\cos(1.2k) + w_k \\ y_k &= x_k^2 / 20 + v_k . \end{aligned} \tag{6.15}$$

Figure 6.3 shows a comparison of the Particle Filter uncertainty bounds observer (Table 6.2) and corresponding EKF algorithm (Table 6.1) performance for the system in Equation 6.13. Although perfect prediction is not achieved for the mean of the data, the PF manages to bound its uncertainty perfectly whereas the EKF bounds only approximately 50% of the data correctly. The number of particles is 500 for the PF. For both algorithms, the initial state is $x_0 \sim N(0,2)$ (i.e. the initial state is normally distributed

with zero mean and a variance of 2), and the measurement and process noise are $v_k \sim N(0,$ 1) and $w_k \sim N(0,10)$, respectively. The signal power, Figure 6.3, is given by

$$P_s = \frac{1}{N}\sum_{i=1}^{N} x_k^2 \,. \tag{6.16}$$

The tracking error is the rms error between the actual and the estimated state (Figure 6.4).

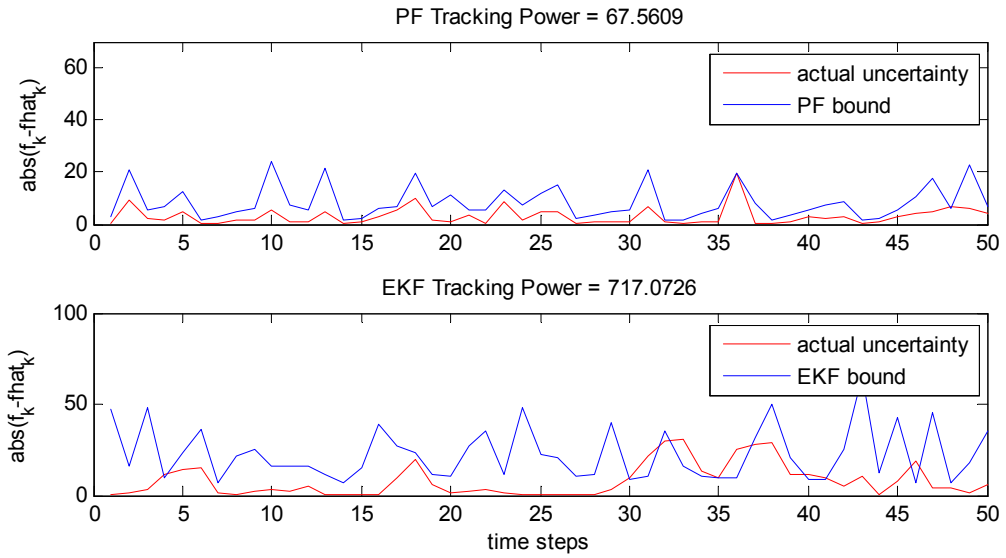

Figure 6.3. Comparison of the Particle Filter uncertainty state and bounds observer (Table 6.2) and EKF (Table 6.1) performance for the system in Equation 5.13. The number of particles is 500 for the PF. For both algorithms, the initial state is x0 ~ N(0,2), and the measurement and process noise are vk ~ N(0, 1) and wk ~ N(0,10), respectively. The figure shows the actual uncertainty with the uncertainty bounds estimates from both the Particle Filter and Extended Kalman Filter techniques.

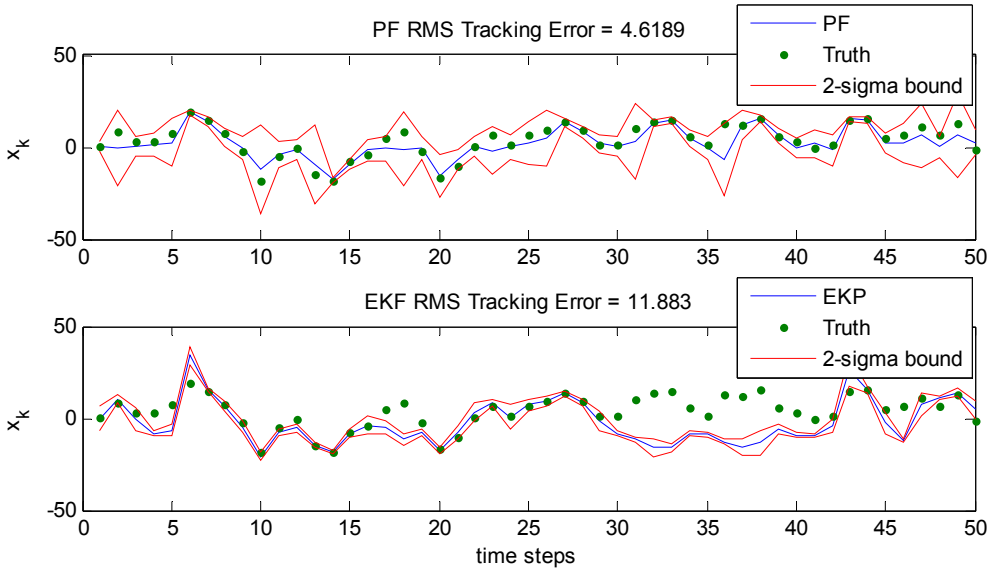

Figure 6.4. Comparison of the PF predictor (Table 6.2) and EKF (Table 6.1) performance for the system in Equation 6.13. In this figure, the actual and estimated state are shown along with the 2-sigma uncertainty bounds estimates.

Particle Filter State and Bounds Estimation using a Sampled-data System Representation

The EKF algorithm requires an analytic expression for the process and measurement models in order to compute and update the Kalman gain (K in Table 6.1). A useful feature of the PF approach is its ability to utilize a sampled-data dynamics representation in the hyperstate propagation equations (Table 6.2). Frequently, all or a

portion of the process dynamics must be estimated using non-parametric techniques such as function regression by either classical or intelligent means, e.g., neural networks [Fernandez and Buckner], [Buckner, Fernandez, Masada]. However, when this is done, care must be taken in order to fuse the prediction estimate with the noisy measurements. A method for sequentially fusing noisy measurements with state estimates subject to a sampled-data function representation is the PF. To illustrate the approach let a sampled-data representation for a portion of the plant dynamics in Equation 6.15 be generated off-line by samples from the function

$$f\left(x_k\right) = (0.5+\varepsilon)\cdot x_k + (25+\gamma)\cdot x_k \Big/ \left(1+x_k{}^2\right) \tag{6.17}$$

where $-100 \le x_k \le 100$, $\varepsilon \sim N\left(0,\sigma_\varepsilon^2\right)$ and $\gamma \sim N\left(0,\sigma_\gamma^2\right)$. This simulates the most basic type of a non-parametric system identification representation which is simply a table of uncertain plant dynamics sampled throughout the operational state space. A physical realization of this might correspond to exercising an RUV through the entirety of its yaw envelope while recording its yaw response. The discrete time sequence of yaw angles would comprise the table entries with the columns corresponding to the yaw variable at time t_k and the rows to possible values of the next state at time t_{k+1}. Figure 6.5 shows a single realization of Equation 6.17 and gives the basic structure of the simple table representation of the uncertain plant dynamics. During the on-line implementation of the PF the particles are propagated using the expression

$$x_{k+1} = f_{k+1} + 8\cos(1.2k) + w_k \tag{6.18}$$

where f_{k+1} is a random sample from the table containing samples of the uncertain plant dynamics. Table 6.3 lists the algorithm used, which to the author's knowledge, is a novel implementation of the basic particle filter algorithm since it incorporates a sampled data process representation for propagating the particles. The basic assumption is that the dynamics are identified offline and stored in a table, neural net, or some other type of non-analytical representation. The offline sample-data process model represents a discrete approximation to the random variable $\hat{x}$ which can be sampled at run-time to evaluate the propagation of each particle through the model dynamics.

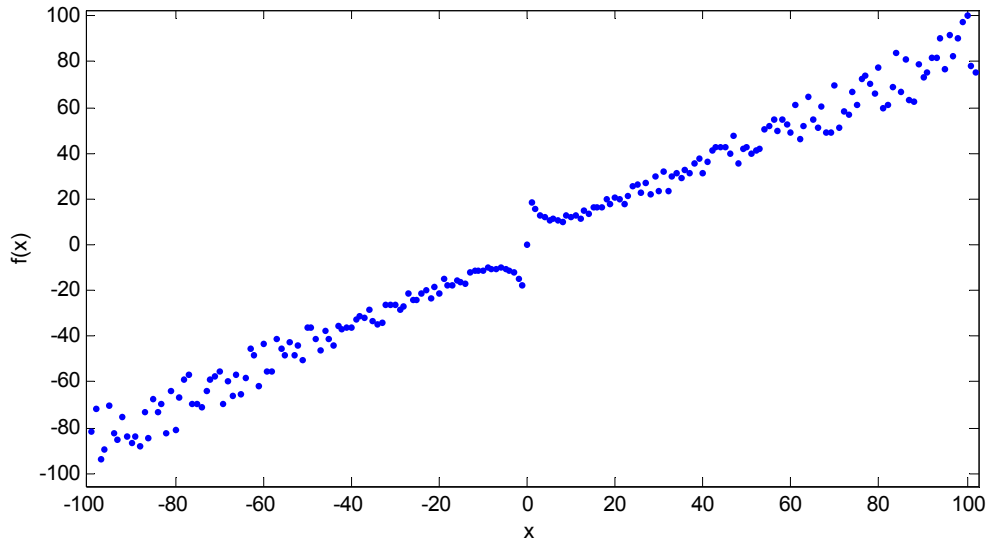

Figure 6.5. Example of a sampled data function representation for use with the PF. The data represent samples of the uncertain process dynamics gathered off-line through experimental procedures. The model, in this case represented as a simple table of data with columns as states at time k and rows as states at time k+1, can be used with the PF state and gain observer to appropriately fuse uncertain process dynamics (sampled data) with noisy measurements to obtain model uncertainty bounds.

Table 6.3. The Particle Filter State and Gain Observer with Sampled Process Dynamics

Initialization:	
$\dot{x}(t) = \hat{f}(x(t)) + \hat{g}(x(t))U(t) + w(t)$ $y(t) = h(x) + v(t)$	Process and Measurement model
$\overline{w}_k = 0,\ E[w_k w_j^T] = Q\delta_{kj}$ $\overline{v}_k = 0,\ E[v_k v_j^T] = R\delta_{kj}$ $E[v_k w_j^T] = 0, \quad E[x_k w_j^T] = 0, \quad E[x_k v_j^T] = 0$	Process and Measurement noise model assumptions
**The Particle Filter PDF X_{PDF} is initially approximated by N random samples normally distributed about the first noisy state measurement.	Initial state and model uncertainty estimate
Measurement Update at time k:	
For each particle:	
Compute the residual Compute the likelihood Normalize the resulting distribution Resample the distribution	Each of these procedures is discussed in detail in [Gordon, et. al.]
Compute the mean: $\hat{x}_k = \overline{X}_{PF}$	Updated state estimate: $\hat{x}_k$ **sent to controller for state feedback**
Prediction Step at time k:	
For each particle:	
Integrate the system equations over the time step The set of final states approximates the PDF	
Compute the variance: $\hat{P}_f = Cov(X_{PF})$	Optimal one step ahead model uncertainty prediction step: $\hat{P}_f$ **sent to controller for gain estimate**

Figure 6.6 shows the results of the PF with uncertain process dynamics simulation. Clearly, the uncertainty bounds increase with increasing process uncertainty. The plots are similar to Figure 6.3. However, now there is process and state uncertainty, hence the increased bounds, increasing the maximum magnitude of PF bound from a value of 22 in Figure 6.3 to 40 in Figure 6.6.

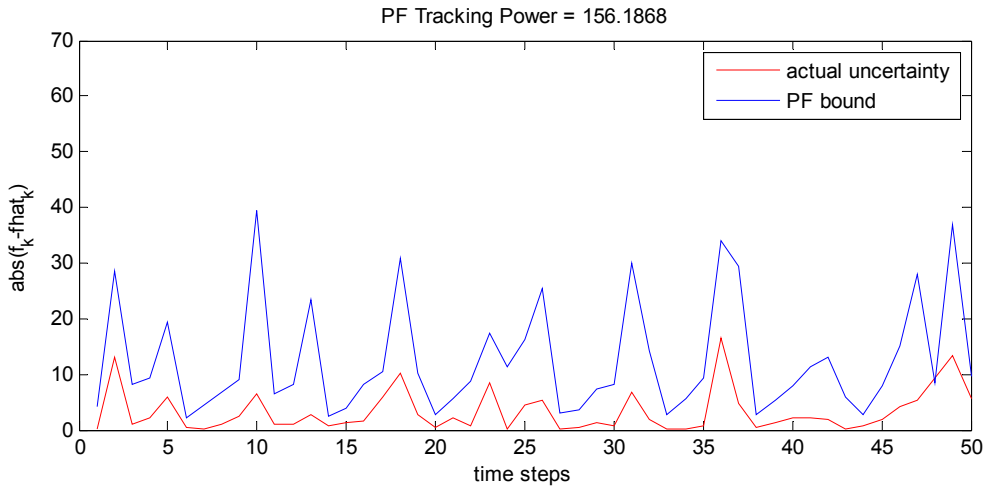

Figure 6.6. Actual uncertainty bounds for the EKF and PF functional uncertainty estimators. The added functional uncertainty in the form of a sampled system accounts for the higher uncertainty bound estimate.

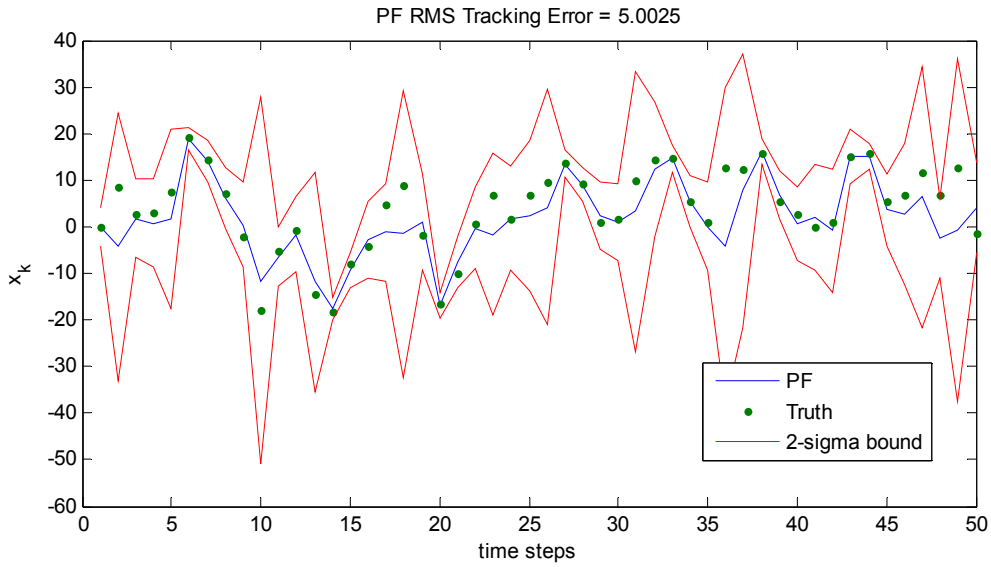

Figure 6.7. The SMCP appropriately fuses sampled process dynamics with noisy measurements to obtain model uncertainty bounds.

OPTIMALLY-ROBUST CONTROL OF ROBOTIC UNDERWATER VEHICLE

Overview

This section details the application of the optimally-robust nonlinear control technique to the control of a class of robotic underwater vehicles. Two approaches to estimating the optimal gain are developed and compared along with a non-optimal constant performance term gain. Optimally-robust control utilizes a stochastic filter to fuse model predictions with noisy measurements to obtain both state and model uncertainty estimates. The state estimate ($\hat{x}_k$ in Tables 6.1-6.2) is utilized for state feedback in the nonlinear controller while the model uncertainty estimate ($\hat{P}_f$ in Tables 6.1-6.2) is utilized in dynamically adjusting the performance term gain (e.g. Figure 6.3). The magnitude of this gain determines the speed of the RUV's closed-loop response and is directly proportional to the vehicle's power consumption in the form of actuator-induced power drain for horizontal/vertical control and thruster cycling in the case of surge speed control. In the simulations to follow attention is limited to the optimally-robust control of the heading subsystem only. However the technique is applicable to any combination of subsystem control configurations. The depth subsystem utilizes a constant gain sliding mode controller. Figure 6.8 shows the optimally-robust controller block diagram.

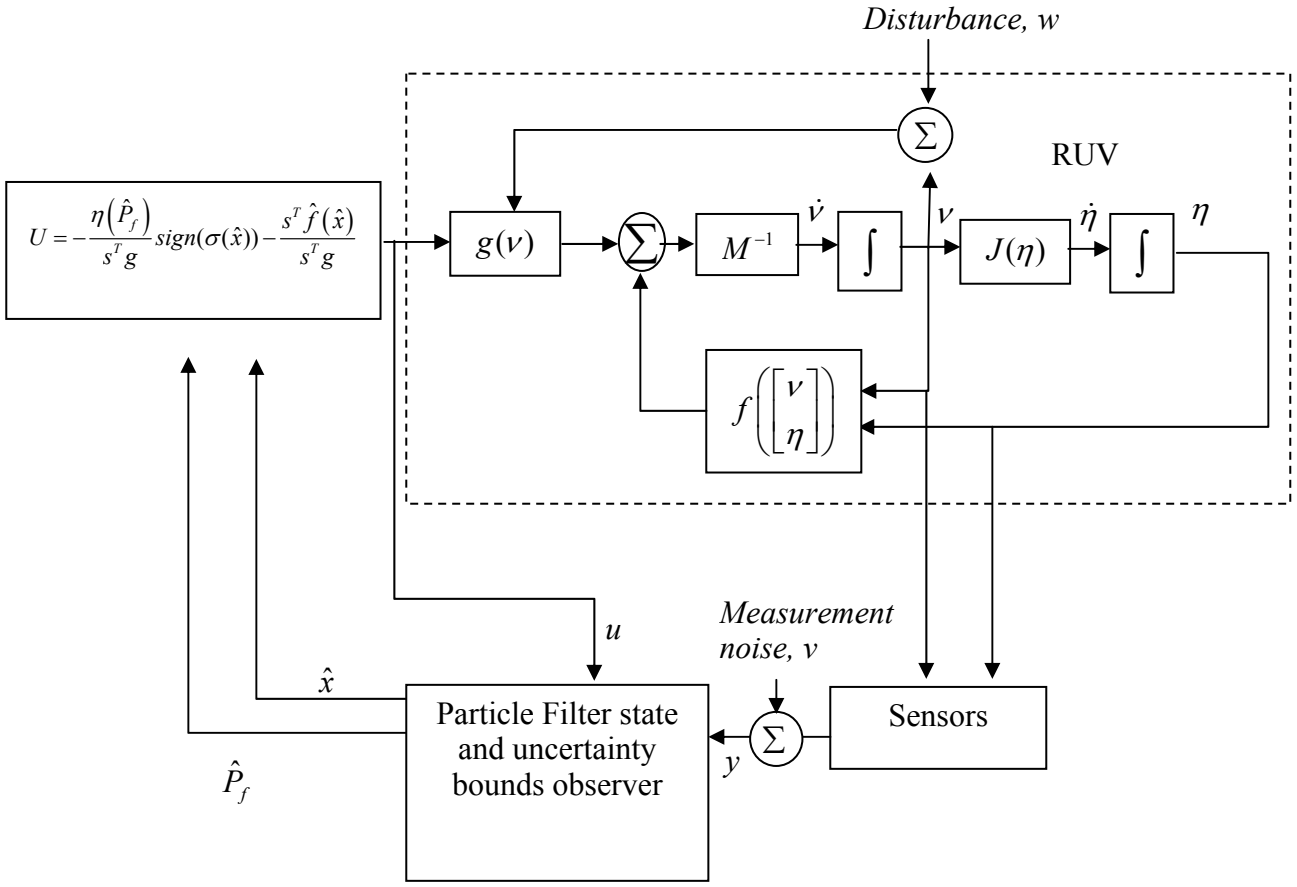

Figure 6.8. Block diagram of the closed-loop RUV with the Particle Filter (PF) optimally-robust control implementation. Noisy sensor data, y, and the control input, u, are used by the Particle Filter to estimate the RUV state and model uncertainty, x and P_f, respectively. These variables are then used in the sliding mode controller to generate the control signal, u. The control and disturbance, , excite the RUV model (dashed box).

Disturbance Model

This section describes the development of a simulated wave disturbance model. Wave disturbance rejection is a practical application of optimally-robust nonlinear control methods. The technique is particularly well-suited to the present RUV class where the fin configuration is such that it cannot independently actuate in a direction perpendicular to its surge component. Since this type of vehicle is frequently used for survey type missions in wave perturbed shallow water and the RUV's instrument suite can include search sensors requiring vehicle stability in yaw such as a side-look sonar [Peterson, et. al.],[Roup and Humphreys], it becomes critical for the controller to reject wave perturbation at all orientations.

The relative fluid velocity influences the vehicle dynamics. Theoretically a vehicle-installed Doppler velocity log (DVL) acoustic current profiler can be used to measure the relative fluid velocity. However, complications exist which limit the accuracy of both horizontal and vertical velocity measurements, particularly in wave-perturbed shallow water where surface wave chop and entrained bubbles exist [Fong and Jones]. Therefore in the following simulations it is assumed that only noisy ground speed measurements are available. This corresponds to the physical situation where the RUV only has a downward mounted DVL for acquiring bottom-locked velocity measurements.

Furthermore shallow water wave models useful for embedded applications have limited fidelity [Peterson, et. al.]. Thus the relative fluid velocity used in the controller modeling equations contains uncertainty. However, by using an approximate wave model, the robust nonlinear controller of Chapter 4 can estimate the wave perturbation,

allowing it to be rejected. At the same time, the controller gain needs to be adjusted according to the RUV's orientation within the wave field because it requires more gain in directions perpendicular to the wave motion due to the lack of independent actuation in that direction (Figure 6.9). Less control power is required to maintain a heading parallel to the wave velocity vector. An optimally-robust heading controller is useful because it improves performance by estimating the orientation-dependent perturbation and adjusting the sliding mode controller gain accordingly. Recall that for guaranteed stability the controller gain must satisfy the state- (and possibly time-) dependent bound given in Equation 6.14. In this section the optimally-robust control methodology is applied to the heading subsystem controller developed in Chapter 4 for the heading control of the full nonlinear simulated RUV system of Chapter 3.

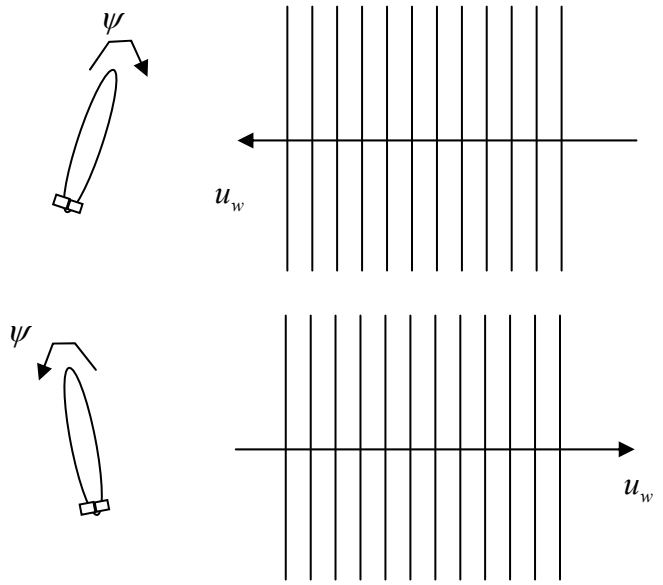

Figure 6.9. Heading oscillation caused by the RUV attempting to maintain a zero sway speed.

The RUV equations of motion, Equation 3.46, can be modified to include the effects of a relative fluid velocity, $\nu_r = \nu - \nu_w$, due to the unmeasurable wave-induced velocity disturbance, ν_w [Fossen]. Assuming the body-fixed wave velocity is slowly varying such that $\dot{\nu}_w \cong 0$, the modified equations of motion are

$$\begin{aligned} M\dot{\nu} &= f(\eta, \nu_r) + g(\nu_r)U \\ \dot{\eta} &= J(\eta)\nu \end{aligned} . \tag{6.19}$$

The wave's body-fixed velocity, ν_w, is related to the earth-fixed velocity, ν_w^e, through the velocity transform

$$\nu_w = J_1^T(\eta_2) \cdot \nu_w^e \tag{6.20}$$

where

$$J_1(\eta_2) \equiv \begin{bmatrix} \cos\psi\cos\theta & -\sin\psi\cos\phi + \cos\psi\sin\theta\sin\phi & \sin\psi\sin\phi + \cos\psi\cos\phi\sin\theta \\ \sin\psi\cos\theta & \cos\psi\cos\phi + \sin\phi\sin\theta\sin\psi & -\cos\psi\sin\phi + \sin\theta\sin\psi\cos\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix}. \tag{6.21}$$

Figure 6.10 illustrates the simplified wave field for the case where $v_w^e = w_w^e = 0$.

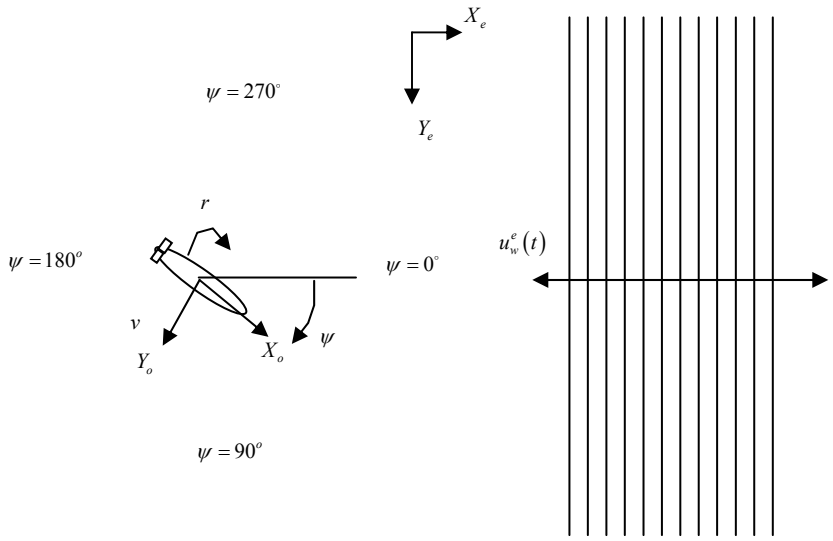

Figure. 6.10. The simplified wave disturbance. Earth- and body-fixed RUV coordinates are shown along with the heading-subsystem state variables as well as each state variable's positive direction.

In this situation one can approximate the earth-fixed wave motion by the equation

$$u_w^e(t) = u_{w0}^e \cos(\omega_w t + \beta_w) \tag{6.22}$$

where u_{w0}^e is the wave disturbance maximum speed, ω_w is the wave frequency, and β_w is the wave field phase. Typically it is not difficult to estimate the approximate direction, wave frequency and amplitude in the RUV mission area prior to deployment. However, estimating the phase is difficult and provides a source of modeling error. This corresponds to structured uncertainty provided by the wave disturbance. This uncertainty

creates a nonzero sway velocity when the vehicle is perpendicular to the wave field direction. The heading subsystem equations are

$$f_h = \begin{bmatrix} \dfrac{1}{m - Y_{\dot{v}}}\left\{-m(u_0 r - y_g r^2) + Y_{HS} + Y_{v|v|} v_r \left|v_r\right| + Y_{r|r|} r \left|r\right| + Y_{uv} u_0 v_r\right\} \\ \dfrac{1}{I_z - N_{\dot{r}}}\left\{-m(x_g u_0 r + y_g v_r r) + N_{HS} + N_{v|v|} v_r \left|v_r\right| + N_{r|r|} r \left|r\right| + N_{uv} u_0 v_r + + Y_{ur} u_0 r\right\} \\ r \end{bmatrix} \quad (6.23)$$

where $\begin{bmatrix} \nu_0 \\ \eta_0 \end{bmatrix} = \begin{bmatrix} u_0 & v_r & 0 & 0 & 0 & r & 0 & 0 & 0 & 0 & 0 & \psi \end{bmatrix}^T$ is the nominal state and v_r is the relative sway fluid velocity. Thus, uncertainty in the wave model impacts the RUV dynamics via v_r. The surge speed is approximated with the nominal speed u_0 for all directions and therefore provides another source of modeling error.

To summarize, the following simulation results utilize the full nonlinear relative velocity RUV equations. These are Equations 6.19 where the wave phase in Equation 6.22 is set to zero, $\beta_w = 0$. The state and gain estimators in the simulated controller, however, utilize a wave model with $\beta_w = 25^0$. As mentioned this results in uncertainty in the dynamics via v_r. The measurements received by the state and gain estimators are not only corrupted by white noise, they do not include measurement of the wave velocity. Specifically they only receive a measurement of the RUV absolute velocity, v.

RESULTS

This section gives the results of the closed loop tracking performance of the optimally-robust control method using the Particle Filter state and gain observer developed in the previous sections. Results from a constant gain and EKF method are shown for comparison. The subsystem control approach developed in Chapter 4 is applied to the heading and depth subsystems with open loop speed control. However, only the heading subsystem utilizes the optimal estimates from the EKF and PF algorithms. The RUV depth tracks a sinusoid pattern between 2 and 4 meters and is maintained by the constant gain control law developed in Chapter 4. Both the system and the model utilize the same parameters for the REMUS RUV. Thus, no parametric uncertainty exists. The only modeling uncertainty comes from the phase, β_w, in Equation 6.22, where the discrepancy between the actual wave phase and the model is 25°. It is assumed that the subsystem state is measurable, i.e., noisy measurements of the heading, sway speed and yaw rate are available for using in the PF and EKF estimation algorithms.

Sway speed with respect to the ground is measured with a DVL., the yaw rate can be measured with a gyro contained in the RUV's Inertial Measurement Unit (IMU), and the heading is typically measured with a magnetic compass [Grenon, et. al.]. The measurement noise is modeled as a zero mean Gaussian process with the following standard deviations [Grenon et. al.]

$$r_v = .03\,m/s, \quad r_r = 1\,\text{deg/sec}, \quad r_\psi = 1\,\text{deg}$$

where the yaw rate standard deviation is quoted for a solid state gyro (e.g. the variety described in Chapter 5). A ring laser gyro would be considerably more accurate (on the order of 10^{-7} deg/sec) [Grenon, et. al.], however the solid state gyro is considerably less expensive and its specification provides a conservative noise estimate. The process noise is modeled as zero mean Gaussian process with standard deviations

$$q_v = .02\,m/s, \quad q_r = 1.14\,\text{deg/sec}, \quad q_\psi = 1.14\,\text{deg}.$$

Furthermore, the process noise is modulated with the current best estimate of the heading, $\hat{\psi}$. That is

$$q(\hat{\psi}) = \begin{bmatrix} q_v & q_r & q_\psi \end{bmatrix} \cdot \left(0.2 + abs\left(\sin\left(\hat{\psi}\right)\right)\right). \tag{6.24}$$

This is justified by the fact that the influence of the relative fluid velocity on the RUV heading subsystem dynamics is more negligible when the vehicle is parallel or anti-parallel to the predominate wave field velocity vector is ψ = 0° or 180° (Figure 6.10). However, to keep the Particle Filter variance from collapsing, the filter process model is seeded with a small amount of noise in the parallel directions; hence the 0.2 coefficient value.

Performance Baseline

As a performance baseline, Figures 6.11 and 6.12 show the simulated RUV tracking performance in the heading subsystem variables when there is no wave disturbance. The mean functional uncertainty in Figure 6.12 is 17.0. The EKF provides state feedback for the controller. The sliding mode heading controller's performance term gain is constant.

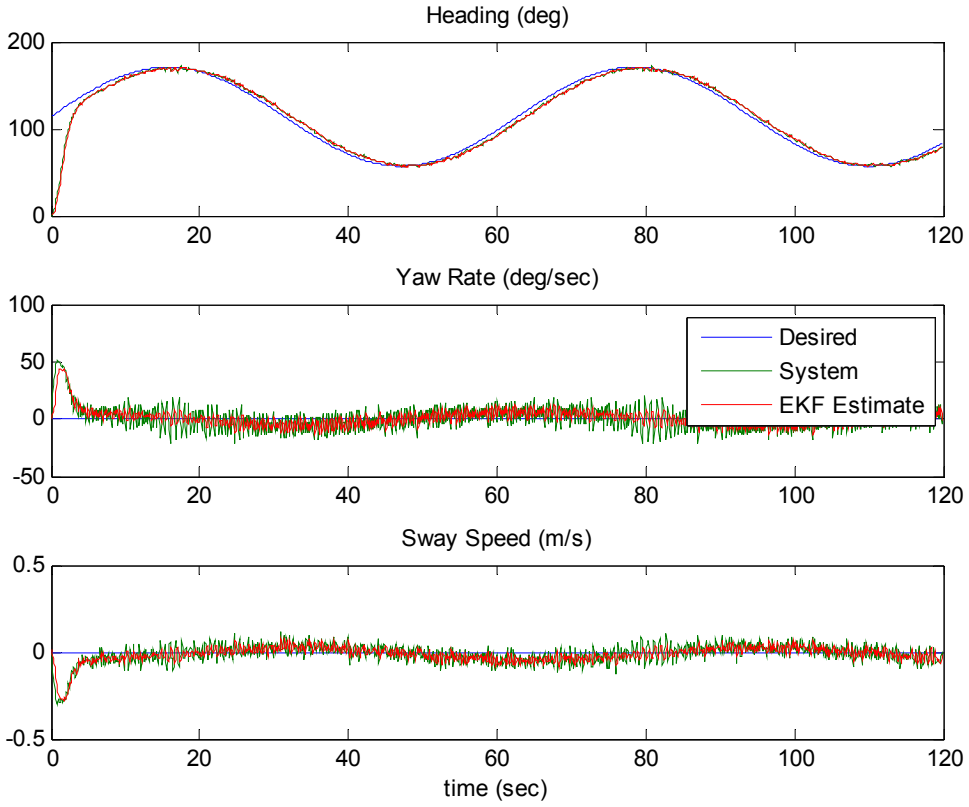

Figure. 6.11. RUV performance baseline. The wave disturbance is turned off in the simulation. The plot shows the heading subsystem state variables.

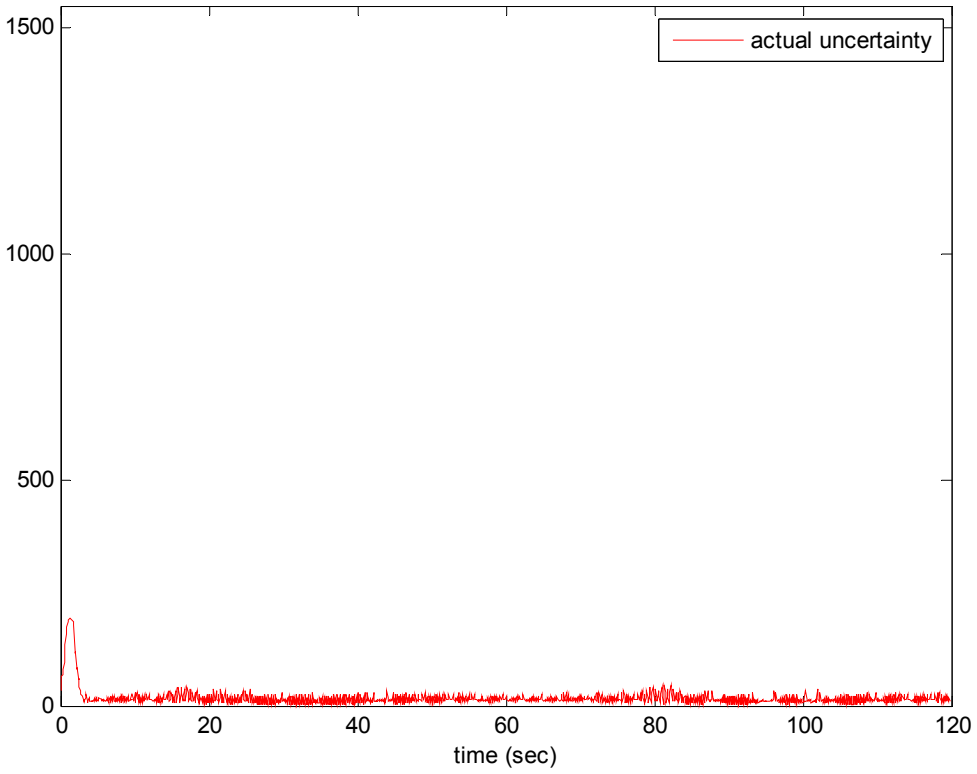

Figure. 6.12. RUV performance baseline: heading subsystem modeling uncertainty when no wave disturbance is present.

Figure 6.13 shows the EKF innovations sequence. This is the sequence of residuals between the measurements and the predictions. The sequence is zero mean and white, suggesting that the filter is performing optimally [Lewis]. This is an often used diagnostic when implementing Kalman filters.

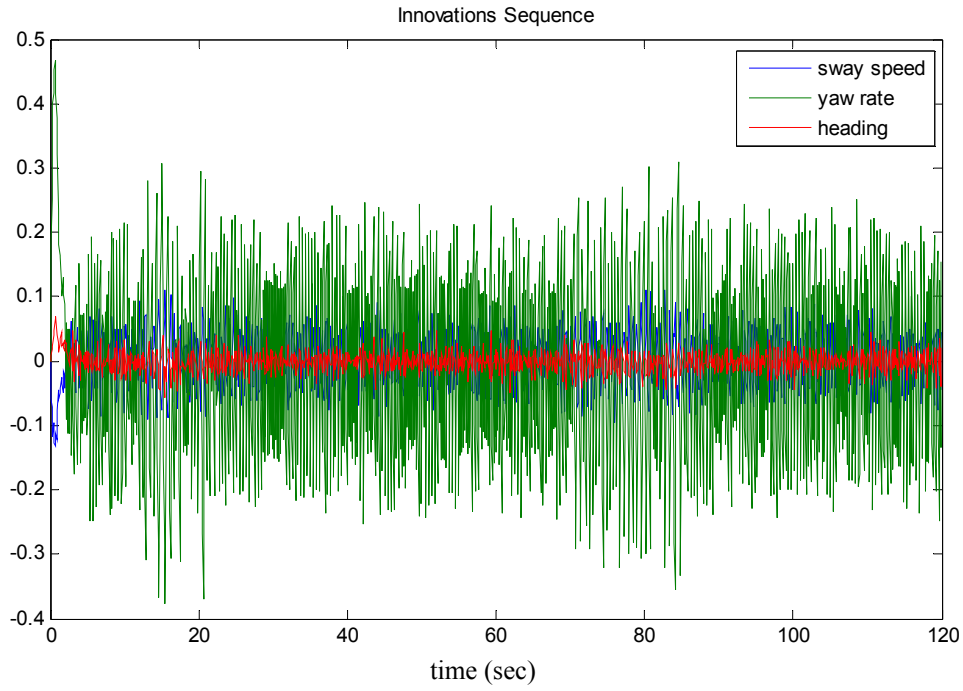

Figure. 6.13. The EKF state estimation innovations sequence: zero disturbance case.

Constant Gain Robust Nonlinear RUV Control

Next, Figures 6.14 and 6.15 show the results for the constant gain bound when the disturbance is activated. The switching gain is

$$\eta_{\Delta f_{\varsigma h}} > \max_{\forall x_h} \left(\left\| \bar{s}_h \right\| \left\| \frac{\partial \Phi_h}{\partial x_h} \right\| \left\| \begin{matrix} \Delta f_{h1} \\ \Delta f_{h2} \\ \Delta f_{h3} \end{matrix} \right\| \right) \tag{6.25}$$

where the max is chosen over all values of the heading substate variables. Again The EKF provides the state estimate for control feedback.

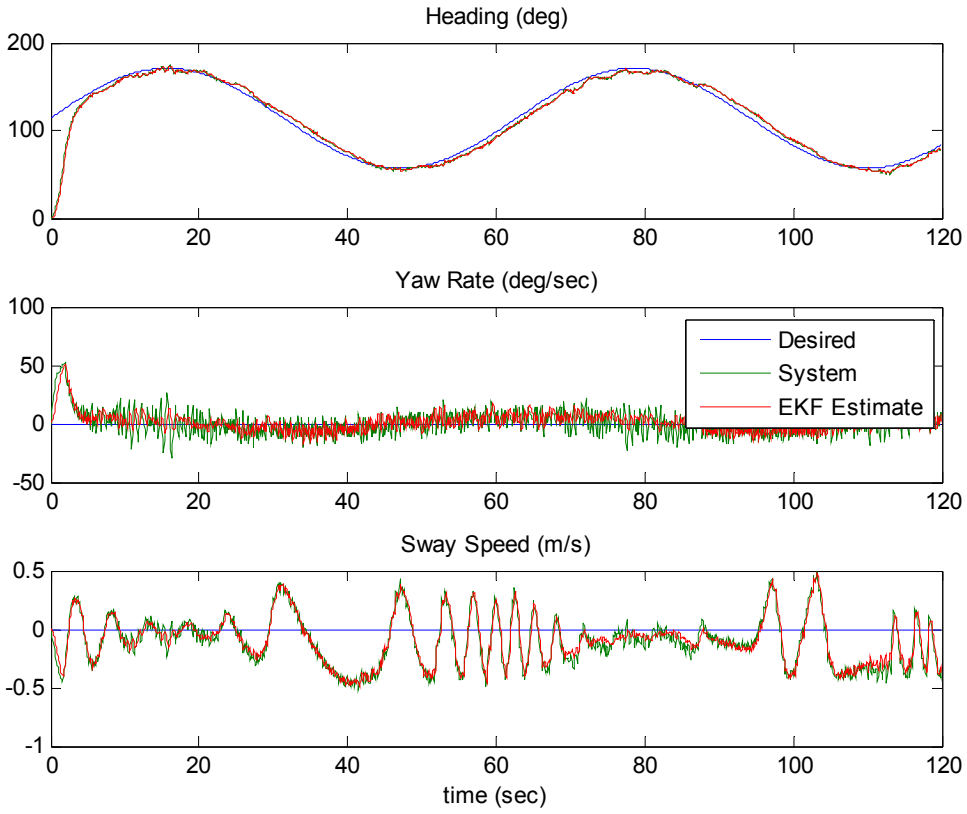

Figure. 6.14. RUV heading subsystem performance for the EKF state estimate and constant gain case.

In each subplot of Figure 6.14 the time between 50 and 80 seconds into the run corresponds to when the vehicle is oriented 90° with respect to the direction of the waves (Figure 6.10).

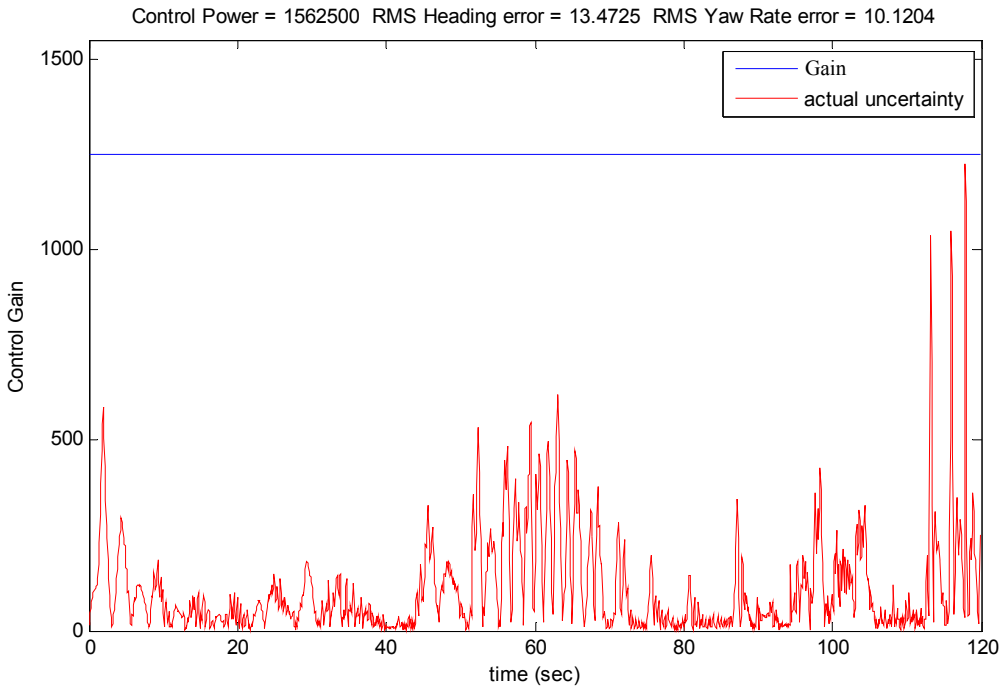

Figure. 6.15. RUV performance for the EKF state estimate and *constant* gain case: heading subsystem modeling uncertainty.

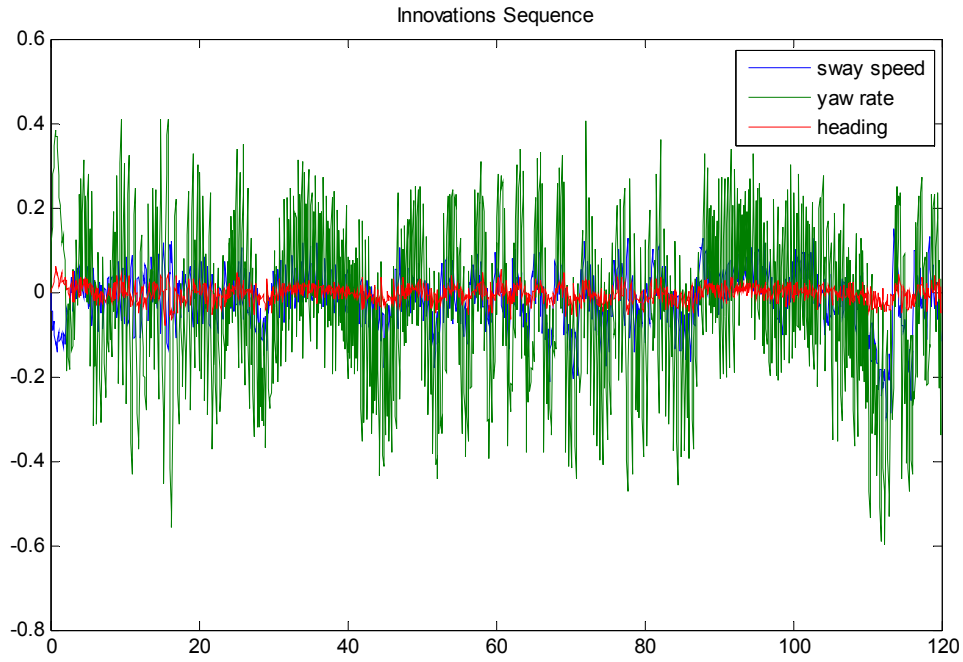

Figure. 6.16. The EKF innovations sequence: constant gain case.

Extended Kalman Filter-based Optimally-Robust Nonlinear RUV Control

Figures 6.17- 6.18 below show the results for the EKF estimated state and uncertainty bound. Clearly, the gain is now time varying due to the state-dependent estimate acquired from the observer.

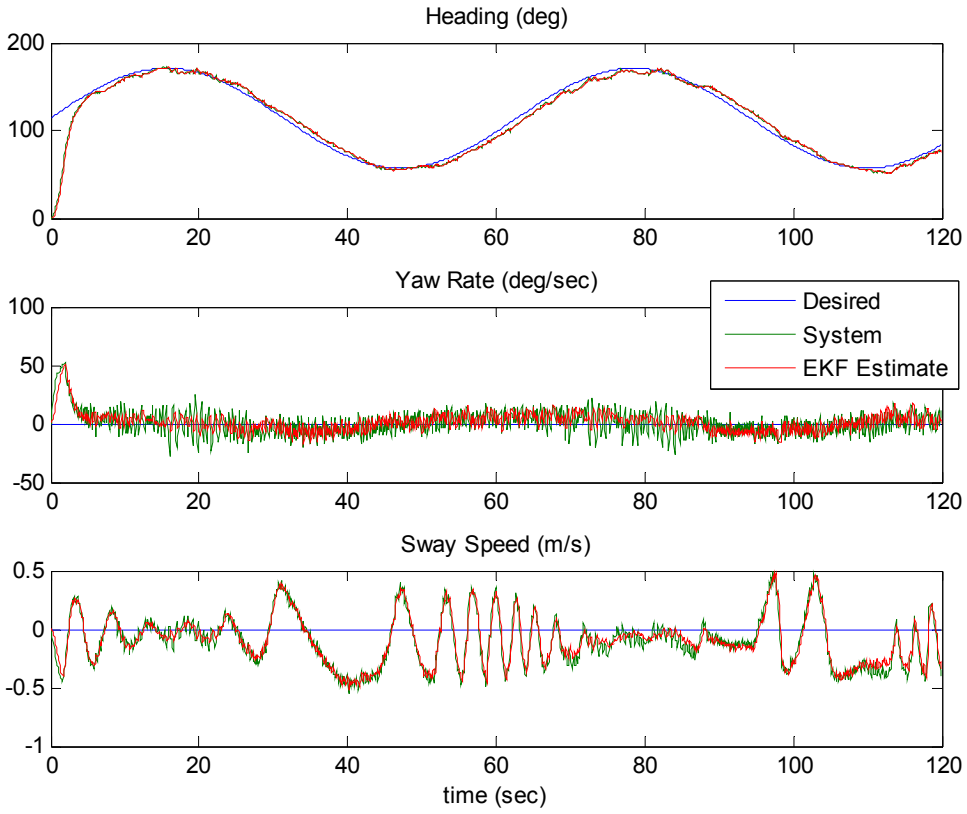

Figure. 6.17. RUV heading subsystem performance for the EKF state and gain estimate.

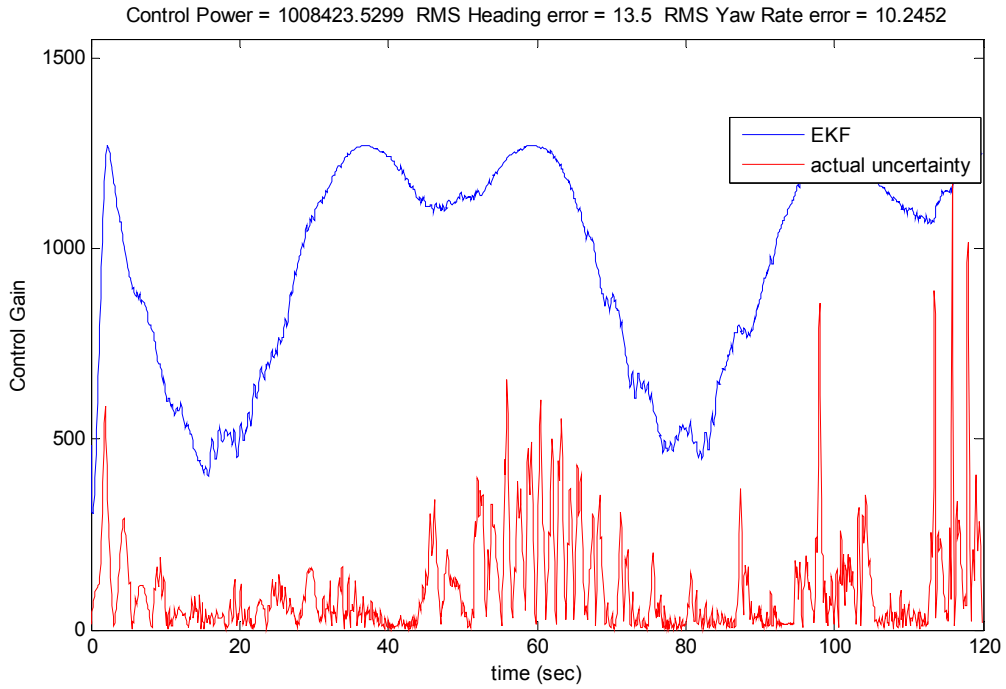

Figure. 6.18. RUV heading subsystem performance for the EKF state and gain estimate: heading subsystem modeling uncertainty.

The EKF state estimate innovations sequence is plotted in Figure 6.19 for each of the heading subsystem state variables. From the data one can conclude that the EKF estimate is colored. However, the filter appears to track the state nicely in Figure 6.19. The effects of the controller convergence could account for the EKF not diverging. However, Figure 6.18 indicates that the error covariance estimate is not very precise. This result is discussed below.

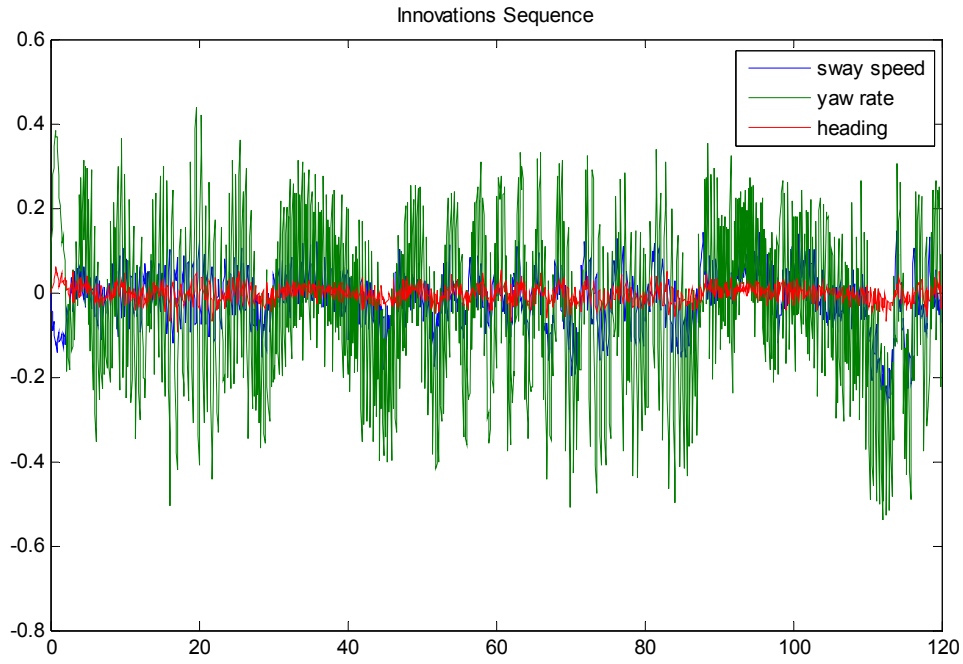

Figure. 6.19. The EKF innovations sequence: EKF state and gain estimation case.

Particle Filter-based Optimally-Robust Nonlinear RUV Control

Figure 6.20 and 6.21 show the results for the PF state and controller gain estimate.

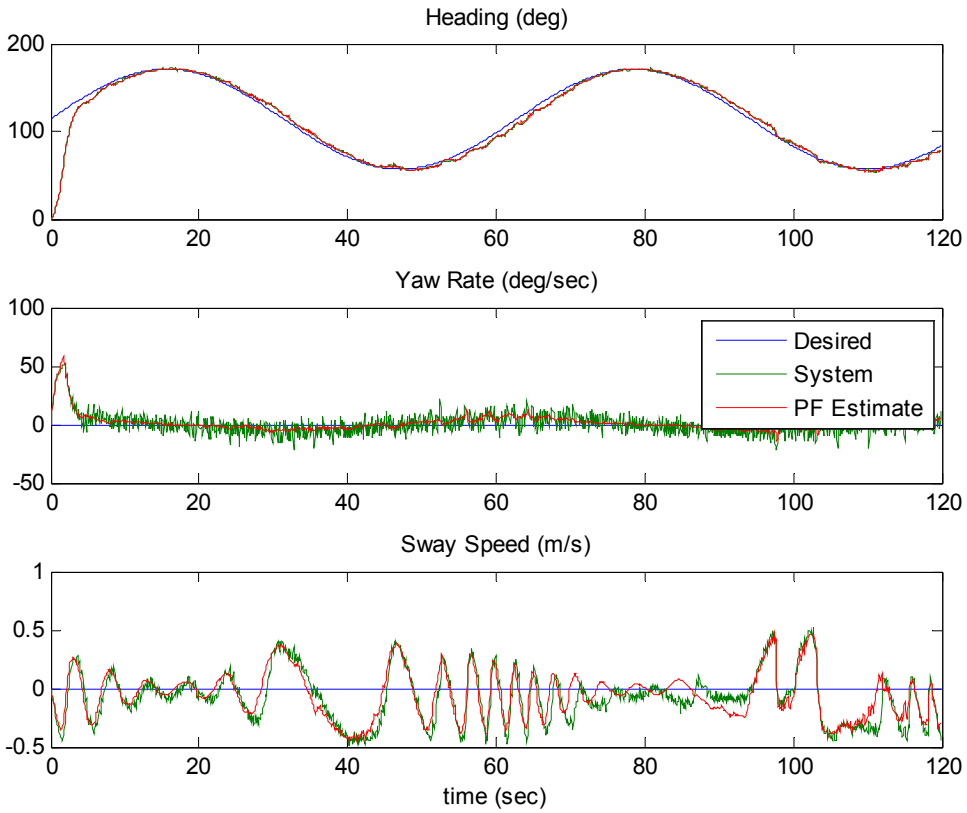

Figure. 6.20. RUV heading subsystem performance for the PF state and gain observer using 10 particles.

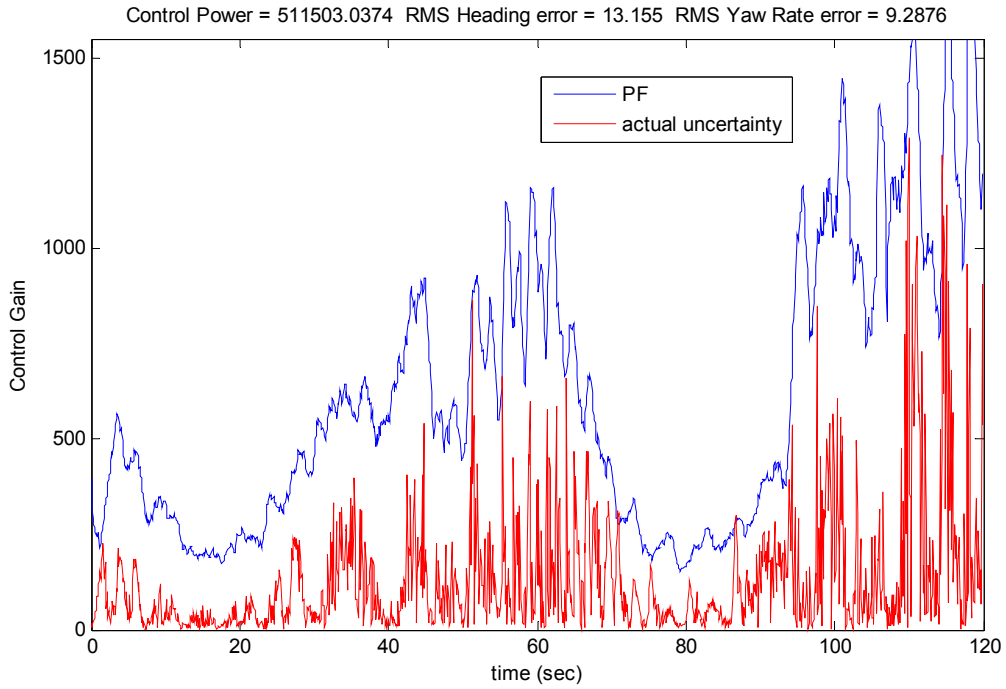

Figure. 6.21. RUV heading subsystem performance for the PF state and gain estimate with 10 particles: heading subsystem modeling uncertainty.

Figure 6.22 shows a plot of the PF state estimate innovations sequence. The subsystem sequences are all zero mean. The heading and yaw rate sequences appear white, however the sway speed shows a color trend.

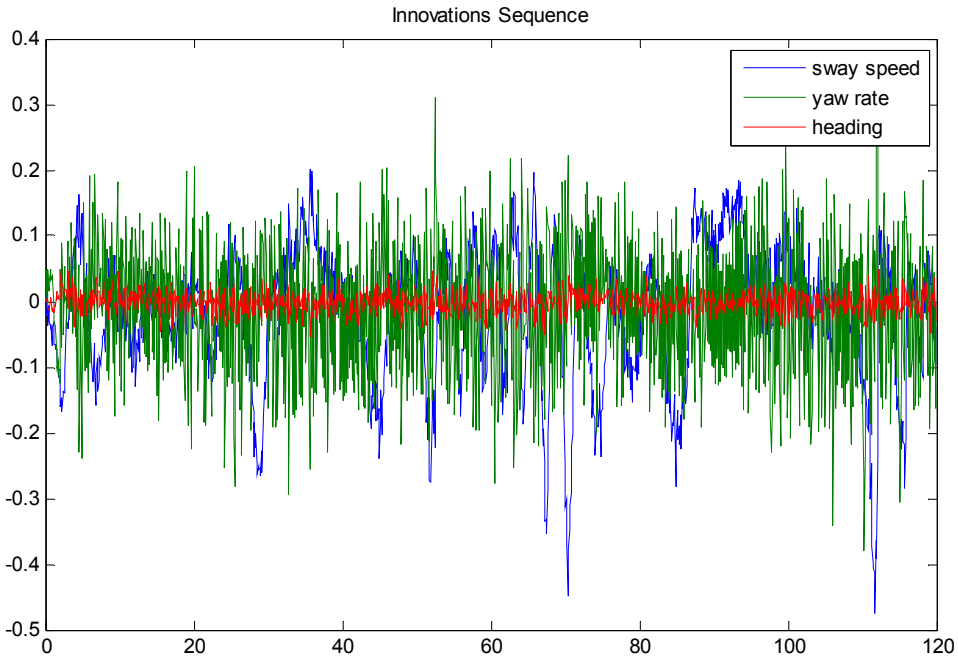

Figure. 6.22. The PF innovations sequence: PF state and gain estimation case with N=10 particles.

Next Figures 6.23 – 6.26 compare the PF-based optimally-robust control method when the number of particles varies.

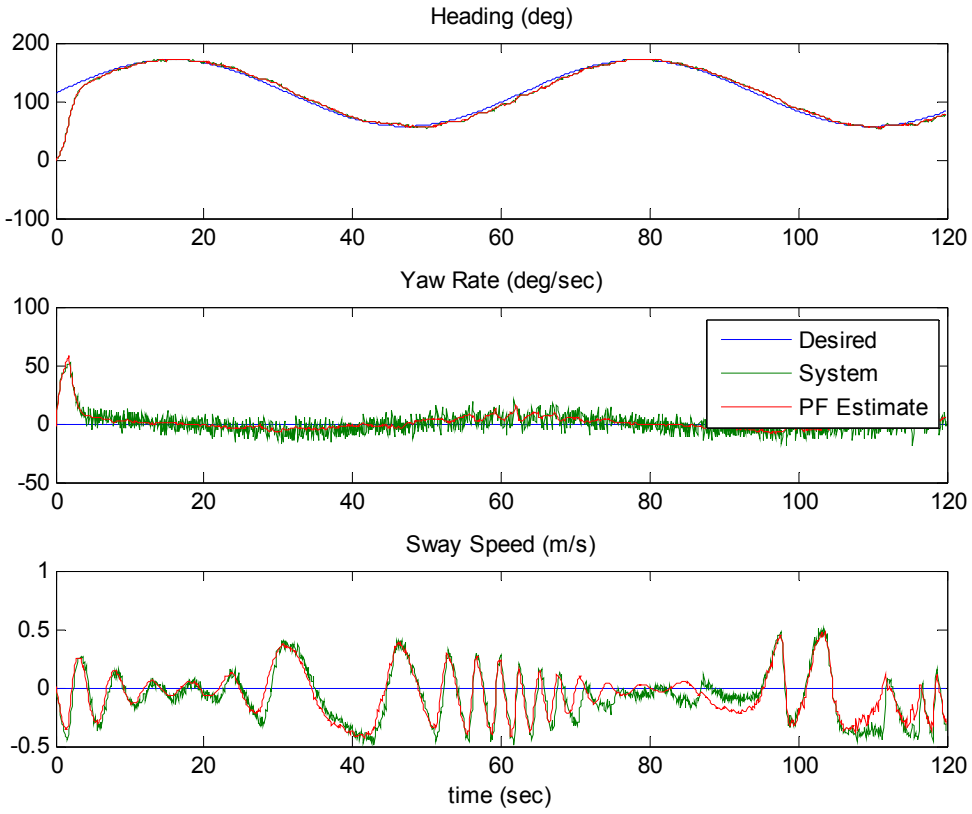

Figure. 6.23. RUV heading subsystem performance for the PF state and gain estimate using 20 particles.

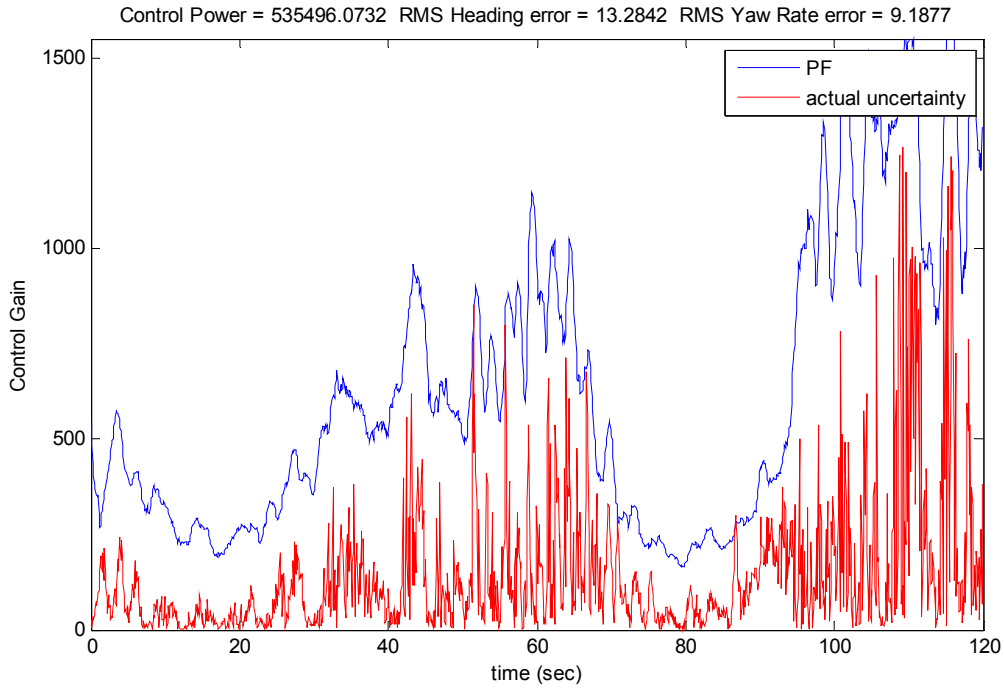

Figure. 6.24. RUV heading subsystem performance for the PF state and gain estimate with 20 particles: heading subsystem modeling uncertainty.

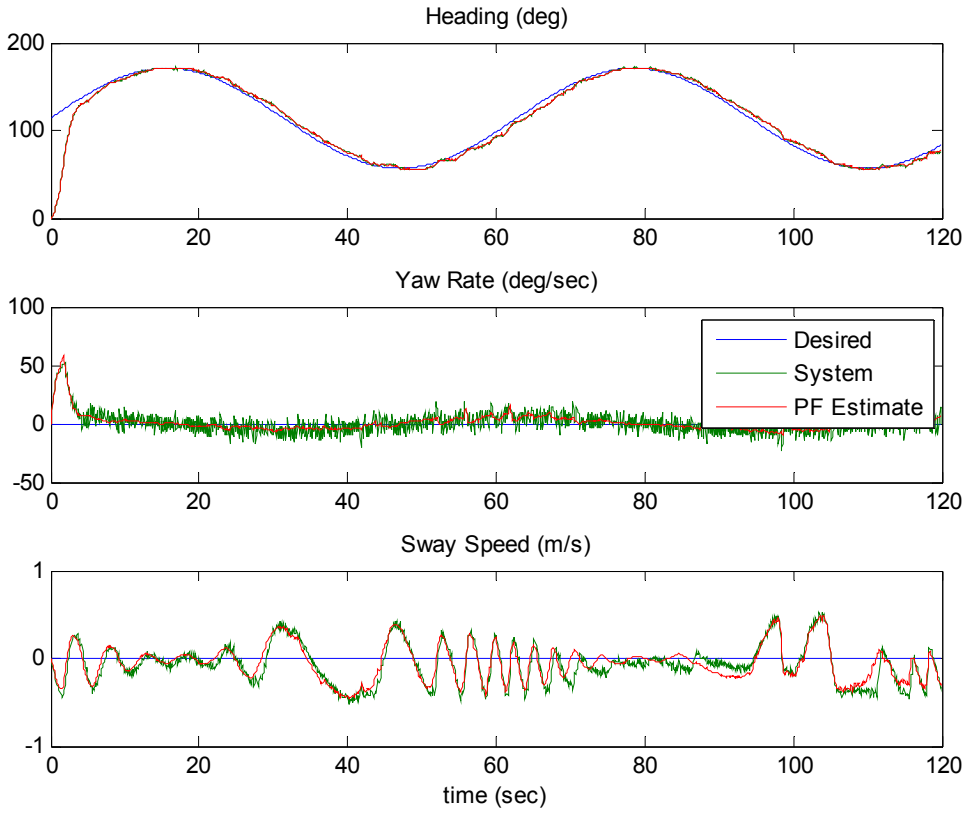

Figure. 6.25. RUV heading subsystem performance for the PF state and gain estimate using 30 particles.

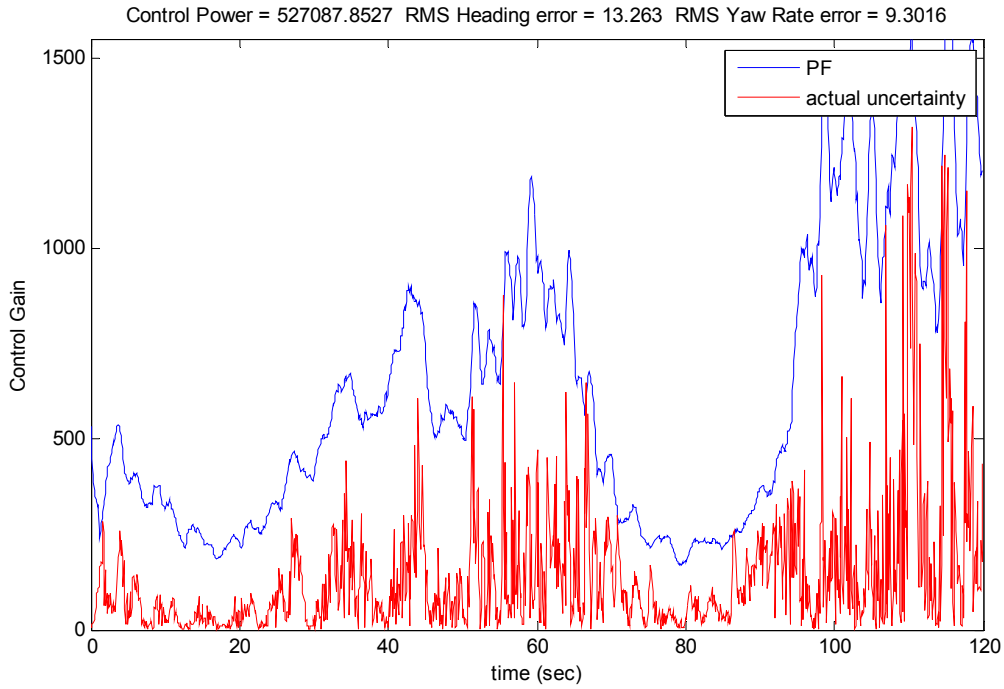

Figure. 6.26. RUV heading subsystem performance for the PF state and gain estimate with 30 particles: heading subsystem modeling uncertainty.

DISCUSSION

The results shown in Figures 6.14-6.22 indicate that the optimally-robust controller utilizes less control power as measured by Equation 6.16 despite process and measurement modeling uncertainty. Furthermore, RMS heading and yaw rate tracking error are decreased over the constant gain and EKF methods. Table 6.4 summarizes the results of simulation for the three methods. The PF result is for the case of N=10 particles. There is a 67% power reduction of the PF method over the constant gain (CG) approach and a corresponding 49.5% reduction over the EKF technique.

Table 6.4. Results Comparison Between the Constant Gain (CG), the Extended Kalman Filter (EKF) and Particle Filter (PF) Optimally-Robust Controllers

	Control Power	Heading RMS	Yaw Rate RMS
CG	1.56×10^6	13.47	10.12
EKF	1.01×10^6	13.50	10.24
PF	0.51×10^6	13.16	9.29
PF-CG Reduction	67.3%	2.3%	8.2%
PF-EKF Reduction	49.5%	2.5%	9.3%

The results for varying the number of particles used in PF algorithm are shown in Figures 6.23 – 6.26. Shows the results for the PF method when the number of particles is modified. Clearly the decrease in yaw rate rms is obviated by the increased per cycle

processing time. Note that 10 particles is the minimum required in this application. With fewer particles the observer becomes unstable.

Table 6.5. Particle Number Results Comparison

Number of Particles	Cycle Time (sec)	Control Power	Heading RMS	Yaw Rate RMS
10	1.3	0.51×10^6	13.16	9.29
20	2.6	0.54×10^6	13.28	9.19
30	4.0	0.53×10^6	13.26	9.30

The EKF heading substate innovations sequences are colored. This is supported by the appearance of the plots in Figure 6.19. The non-whiteness of the sequences suggest that the modeling equations in the EKF are inaccurate. Since there is no parametric uncertainty in the RUV model, the source of error is clearly in the wave velocity model. Despite this uncertainty, which manifests in the non-white innovations sequences of Figure 6.19, the EKF state estimate closely tracks the actual RUV state in Figure 6.17. One way to account for this is to assume the closed-loop control feedback improves the EKF state tracking performance. However, the EKF gain estimate in Figure 6.18 does not track the modeling uncertainty as tightly as the Particle Filter algorithm (Figure 6.21). A partial explanation is as follows. In the implementation of the discrete Extended Kalman Filter (Table 6.1) one must discretize the continuous process noise

covariance matrix, $Q(t)$, to obtain its discrete counterpart, Q_k. The relation between the two is

$$Q_k = \int_{t_{k-1}}^{t_k} \Phi_h\left(t_k,\tau\right) Q\left(t\right) \Phi\ \left(t_k,\tau\right) d\tau \tag{6.26}$$

where Φ_h is the heading subsystem state transition matrix. It is found by integrating from time t_{k-1} to time t_k the differential equation

$$\dot{\Phi}_h = \left(\frac{\partial f_h}{\partial x_h}\right)\Bigg|_{\hat{x}_k}, \quad \Phi_h\left(t_{k-1},t_{k-1}\right) = I\ . \tag{6.27}$$

For the wave disturbance model the continuous process covariance matrix is given by

$$Q\left(t\right) = \begin{bmatrix} q_v^2 & 0 & 0 \\ 0 & q_r^2 & 0 \\ 0 & 0 & q_\psi^2 \end{bmatrix} \left(0.2 + \left|\sin\left(\hat{\psi}\right)\right|\right)^2 . \tag{6.28}$$

Therefore, solving for Q_k at each time step involves integrating Equation 6.26 after solving Equation 6.27. By simply using Equation 6.28 in place of the discrete expression, that is, let $Q_k = Q(t)$, one simplifies the implementation. The approximation introduces errors as Δt_k increases. Since the Particle Filter method explicitly evaluates each particle state using the process equations, no noise covariance matrix is required. This may allow the PF state and gain observer to track the modeling uncertainty more tightly.

Chapter 7. Conclusions and Recommendations

SUMMARY

Chapter 2 of this dissertation showed the development of a rigorous 6 degree of freedom nonlinear mathematical model of a closed loop RUV system and motivated the model veracity by a successful comparison with the measured response of a real vehicle, the RUV REMUS 100. Previous works have focused only on the development and verification of open loop models for the REMUS. Controllers developed in Chapter 4 were applied to the RUV model. Separate depth and heading regular form sliding mode controllers were derived and applied to the full nonlinear RUV system equations. The resulting closed-loop performance was analyzed and a comparison made to the state linearized controller in the case of modeling uncertainty. It was shown that the regular form sliding mode control law outperformed the linearized controller in terms of stability for the depth and heading set-points used in Chapter 3. Chapter 5 described the mechanical, electronic and experimental robotic underwater vehicle and gives the results of the implementation and comparison of both conventional PD heading controller and the regular form sliding mode controller. The purpose of the experimental RUV is to show that the more complicated formulas of the regular form sliding mode controller can still be implemented real-time and that the controller's performance with regard to modeling uncertainty justifies the added complexity. Chapter 6 defined the optimal variable structure control problem and proposes a method for its solution. The technique involves estimating the state varying modeling uncertainty of a nonlinear plant using a

novel single-step-ahead Particle Filter algorithm. It is shown that the algorithm optimally fuses noisy measurements and uncertain plant dynamics. The algorithm performance is demonstrated on an example system with hard nonlinearites. A practical wave disturbance model is developed which results in unmodelled RUV dynamics. The optimally-robust EKF- and PF-based controllers are applied and compared to a constant gain control law. It is shown in the discussion section in Chapter 6 that the RUV closed-loop performance under the Particle Filter-based optimally-robust control minimizes the required control energy as measured by the power in the switching controller's gain signal (the primary component of the control effort), as well as the heading and yaw rate rms levels.

DISSERTATION CONTRIBUTION

The key dissertation contribution is the formulation and solution of the optimally-robust control problem for the class of robotic underwater vehicles characterized generically by inherent heading instability, high fineness ratio, axial symmetry, metacenter stability, and an inconsistent actuator configuration. The approach involves showing that a regular form sliding mode control law can be designed for each of the three fundamental subsystems which will provide RUV tracking and regulation stability and that the controller gains and sliding manifold coefficients can be selected in such a way as to simultaneously provide robust tracking stability and satisfy certain optimality conditions in the presence of disturbances and modeling uncertainty. The technique involves estimating the time varying modeling uncertainty of a nonlinear system using a nonlinear one step ahead state estimation approach utilizing Particle Filtering methods.

A further contribution of this dissertation is the application of the foregoing nonlinear control methodology to the tracking control of a simulated instantiation of a vehicle from the aforementioned class of RUVs, Hydroid's REMUS. To the author's knowledge this presents the first instance of such an application since the canonical RUV control for the REMUS utilizes linear PID methods [Allen et. al. 1997]. The regular form sliding mode control method is implemented on a real RUV and compared to PID-type techniques for the heading subsystem.

A final offering of this dissertation is the rigorous development and verification of a general 6 degree of freedom mathematical model of a closed loop RUV pilot algorithm for the specified class of RUV. Previous works have focused only on the development and qualitative verification of open loop models [Prestero].

RECOMMENDATIONS

Regular Form Sliding Mode Control

The justification for the selection of the hypersurface coefficients in the switching control term remains to be developed. The current technique involves grid searching the 3-dimensional region to find sliding surface coefficients which give the desired RUV closed loop performance. A constructive proof establishing the conditions of regulation and tracking stability of the regular form slow dynamics Equation 4.17 (i.e. the system dynamics confined to the sliding surface) would serve as an algorithm for analytically computing the sliding surface coefficients. However such a proof is unlikely to exist

without serious restrictions. Developing an existence proof and utilizing advanced search algorithms such as genetic algorithms is a more realistic approach.

A further control-related task would be to develop the Single Input Multiple State (SIMS) control law as a basis of comparison along with the PD/PID type controller already developed in Chapter 3. The SIMS method utilizes a variable structure control approach wherein the vehicle dynamics are linearized about an operating point and a performance control term is injected which cancels the remaining nonlinearities. A second control signal is injected that asserts the linear feedback performance. The gain matrix of this term can be computed using pole placement while the sliding surface coefficients are computable from a matrix eigenvalue equation in the dual space of the linearized dynamics [Christi, et. al.].

Finally it would be edifying to investigate the regular form multi-input nonlinear sliding mode controller in an attempt to obviate the subsystem separability assumptions introduced in the Chapter 4 subsytem control laws.

Optimal Gain Selection

Observer-controller stability in the nonlinear case is an expansive and little understood field of theoretical nonlinear control. In the (deterministic or stochastic) linear systems case, the separation principle allows one to independently develop the observer and controller. If each is stable, then the combined observer-controller system is stable [Bishop]. There is no general result for nonlinear systems [Bishop]. It would be

informative to rigorously establish the combined EKF-RUV and PF-RUV stability for the class of vehicles studied in this work.

Application to the RUV Model

The RUV class of this dissertation is optimized for various payloads and sensor configurations. For example the REMUS 100 has radiometric, turbulence measurement, DIDSON, GPS, plankton pump, and bioluminescence configurations, each with a separate hull profile and inertial characteristics [Hydroid]. A practical application of the sliding mode control theory to RUV class of this dissertation is the rejection of model mismatch due to payload and/or sensor configuration changes. Typically before a new sensor can be mounted to the robot, a complete knowledge of the sensor's inertial and drag characteristics must be established. Clearly this may be difficult or impossible to estimate accurately. Furthermore, each time the sensor is removed or added a new set of controller gains are computed by the RUV computer. This introduces the possibility that the incorrect gains are loaded, jeopardizing mission success. It would be instructive to simulate model uncertainty due to payload changes and establish the performance of the robust controller of this dissertation. Another form of state/time varying uncertainty worthy of consideration is the case of dynamically varying hydrodynamic coefficients.

Appendix

EXPERIMENTAL RUV SCHEMATICS

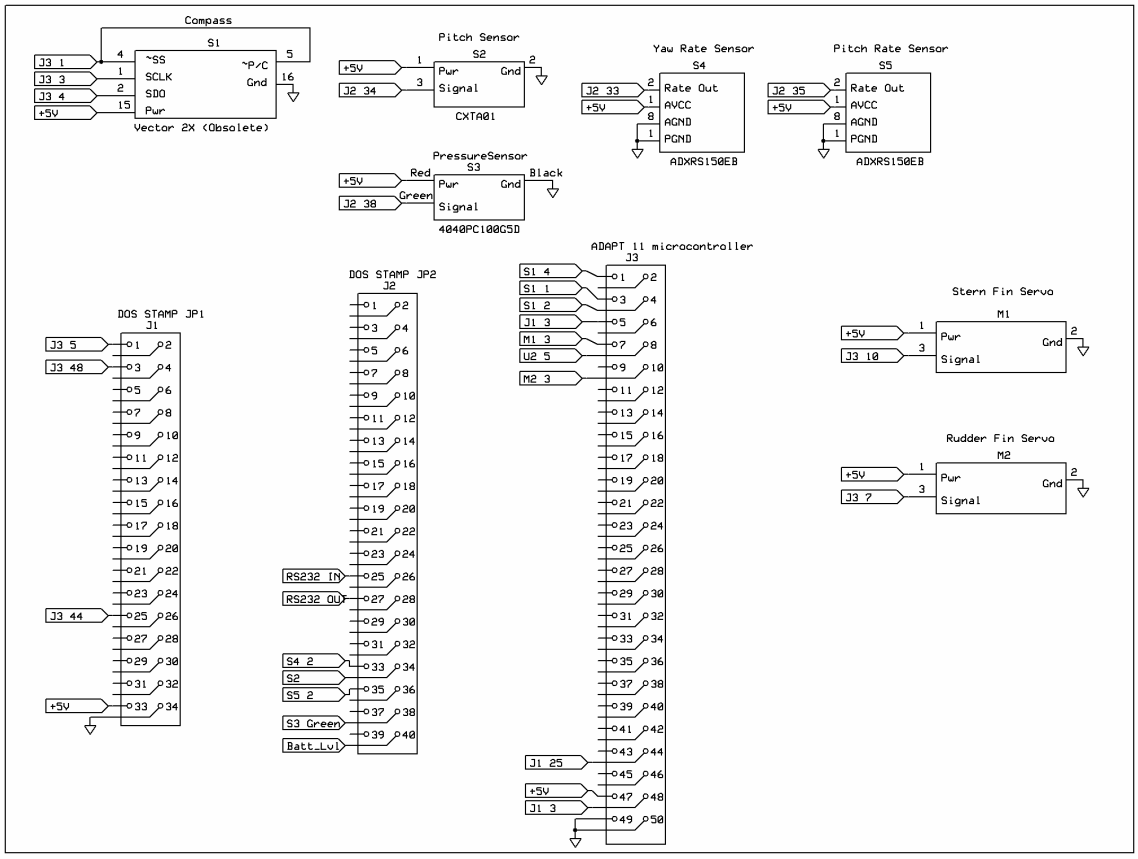

Figure A.1. RUV schematic showing sensor, fin servo and inter-microcontroller connections.

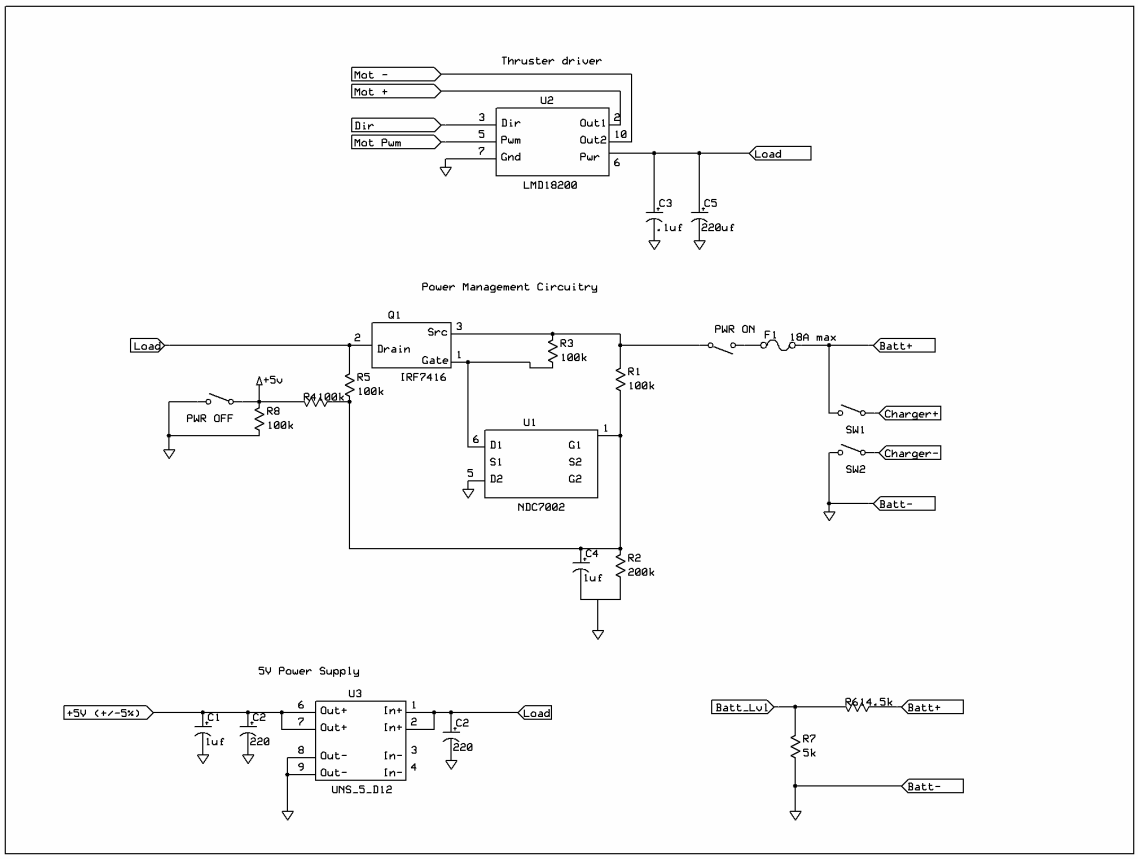

Figure A.2. RUV custom circuit card schematic.

EXPERIMENTAL RUV CODE LISTING

This section describes the code for TESTSM1.C, AUVMODEL.C, ADAPT11.C and the assembly code for the 6811-based Adapt 11 microcontroller. These routines were written by the author. Further code utilized with the RUV, but not written by the author include DSCOMM.C, DSRTC.C, DSADC.C, DOSSTAMP.C. These routines were written by Ivan Baggitt and can be downloaded at www.bagotronix.com. Numerical routines for computing matrix-matrix products, vector-matrix products, matrix inverses, allocating/de-allocating vector and matrix data types, etc. can be found in [Schilling and Harris] in the file NLIB.C.

Executive Mission Code for Dos Stamp: TESTSM1.C

The following code is the main driver code for evaluating and comparing the sliding mode and PID controllers on the experimental RUV developed in Chapter 5. The code runs in a timed loop by reading and executing each line of a mission plan text file, mp.txt. Each line of the file contains a value in seconds for the length of the current mission leg, '`legTIME`', the thruster duty cycle, '`motorDutyCycle`', the desired depth '`depth`', the desired heading '`heading`', and a Boolean value to indicate which controller to use '`USE_SM`', where USE_SM=1 utilizes the sliding mode autopilot, and USE_SM=0 utilizes the PID autopilot. The RUV state (depth, heading, yaw/pitch rates, and pitch) is recorded in a text file four times per second.

```
#include <dos.h>
#include <stdio.h>
#include <string.h>
```

```
#include <float.h>
#include <stdlib.h>
#include <time.h>

#include "c:\borlandc\bin\phd\nlib\tNlib.h"
#include "c:\borlandc\bin\phd\auv\auvModel.h"

#include "c:\borlandc\bin\phd\dosstamp\stdinc.h"
#include "c:\borlandc\bin\phd\dosstamp\dosstamp.h"
#include "c:\borlandc\bin\phd\dosstamp\dsrtc.h"
#include "c:\borlandc\bin\phd\dosstamp\dsadc.h"
#include "c:\borlandc\bin\phd\dosstamp\tdscomm.h"
#include "c:\borlandc\bin\phd\dosstamp\alarm.h"
#include "c:\borlandc\bin\phd\dosstamp\adapt11.h"

//Some convenient global variables
BOOLEAN alarm= FALSE, USE_SM;
long ticks;
float clk_tck = 16.0;//1024; change below RTCsetperiod
FILE *LOGFILENAME;

int  alarmIndex=0;
char *alarmStr[8] = {"NULL","low battery","timer expired"};
float legTIME = 10, turnaroundTIME = 5, TIME = 0, logPeriod = .25;   //  in seconds

int cdecl matherr(struct exception *e){return 1;} //don't print fp NAN errors to screen

void interrupt RTC_ISR (void) {
       inportb (RTC_REGC);           // clear PF flag bit
       ++ticks;
       SEOI(INT3_VEC);
}

int main (void){
       UWORD motorPeriod = 34000, portMotorDirection = 1, stbdMotorDirection = 1, apON =
0, apHeading = 57, apDir = 1;
       UWORD hr, min, sec,  portMotorLine = 0, stbdMotorLine = 0;

       FILE *fp;
       LONG lticks;
       float  motorDutyCycle = .15, depth = 0, heading = 0;
       float gainCorner = 3, deadBand = 1.0;  //in units of feet

       int     i, NumIts=1000;
       float   t = 0;

       _control87 (MCW_EM, MCW_EM);  //mask fp exceptions overflow and div-by-zero
       /*do some initialization stuff for the numerical routines*/
       initRUVModel();
       //zerovec(Xd,totStates);  //use this for testing when comparing matlab and c-code
control code numerics

       /*setup periodic interrupt */
       /*prepare INT3 for activity*/
       lticks = ticks = 0L;
       IRQinit (INT3_VEC, RTC_ISR, 7, EDGE);
       RTCsetperiod(RTC_16HZ);// (RTC_1024HZ);
       RTCalarmcontrol (TRUE, FALSE, FALSE);
       COM2setup();
       adapt11Reset();

       RTCgettime (&hr, &min, &sec);

       LOGFILENAME = fopen("c:\logfile.txt", "w");
       if(!LOGFILENAME){printf("NULL logfile pointer");exit(1);}
       fprintf(LOGFILENAME,"Mission start time:  %02d:%02d:%02d", hr, min, sec);
```

```
        fprintf(LOGFILENAME,"\nBattery level at mission start: %.2f
volts\n\n",getBatteryLevel());


        fp = fopen("c:\mp.txt","r");
        if(!fp){printf("NULL mp file pointer");exit(1);}

        /*initialize stern fin servo*/
        setMotorONLine(1);
        setMotorDutyCycle(1,.087, motorPeriod);
        /*initialize rudder fin servo*/
        setServoDutyCycle(.087, motorPeriod);
        //initialize thruster
        setMotorDirection(0, 1);  //thruster is motor 0
        //setMotorONLine(0);

        unmask (INT3_VEC);   //start timer

        i = 1;
        while(!feof(fp)){
                fscanf(fp, "%f  %f %f %f  ", &legTIME, &motorDutyCycle, &depth, &heading,
&USE_SM);
                Xd_heading[12] = heading/57.0;  //in degrees, convert to radians for
control routines
                Xd_depth[9] = depth;  //in meters
                setMotorDutyCycle(0, motorDutyCycle, motorPeriod);
                setMotorONLine(0);
                alarm = FALSE;
                while(!alarm){
                        TIME = (float)(ticks-lticks)/clk_tck;
                        //if (USE_SM){
                                //autoPilotSM(t, Xd_depth, Xd_heading, motorPeriod);
                                //}
                        //else {
                                autoPilotPID(t,Xd_depth, Xd_heading, motorPeriod);
                                //}

                        if(TIME > legTIME) {
                                alarm = TRUE;
                                lticks = ticks = 0L;
                        }
                        if(TIME > (float)i*logPeriod){

                                fprintf(LOGFILENAME, "%.2f\t%u\t%.3f\t%.2f\t%.3f\t%.2f\n",
                                TIME, getCompassReading() ,getYawRateSensor()
,getPitchSensor(),getPitchRateSensor(),
                                getDepthSensor());
                        autoPilotPID(t,Xd_depth, Xd_heading);
                                i++;
                        }
                }
        }

        fclose(fp);  //close the mission plan file
        setMotorOFFLine(0);
        /*stern fin to zero degrees*/
        setMotorDutyCycle(1,.087, motorPeriod);
        /*rudder fin servo to zero*/
        setServoDutyCycle(.087, motorPeriod);

        /*shut off periodic interrupt (timer)*/
        mask (INT3_VEC);
        RTCalarmcontrol (FALSE, FALSE, FALSE);

        /*Close mission record file*/
        RTCgettime (&hr, &min, &sec);
```

```
        fprintf(LOGFILENAME,"\nalarm condition: %s", alarmStr[alarmIndex]);
        fprintf(LOGFILENAME,"\nMission stop time: %02d:%02d:%02d", hr, min, sec);
        fprintf(LOGFILENAME,"\nLast heading: %u", getCompassReading() );
        fprintf(LOGFILENAME,"\nBattery level at mission stop: %.2f
volts\n",getBatteryLevel());
        fclose(LOGFILENAME);


        return 0;
}
```

RUV Model Code: AUVMODEL.C

The following code calculates the RUV model values for the regular form sliding mode controller. The routines require the numerical libraries, NLIB.C, found in [Schilling and Harris].

```
#include <math.h>
#include "c:\borlandc\bin\phd\auv\auvModel.h"


const int numStates = 6, numInputs = 4, totStates=12;
//max fin values (see RUV notebook p.87)
const float DSMAX = 15.*PI/180.;
const float sternDCMAX = .109;
const float sternDCMIN = .060;
float sternDCslope;
float sternDCintercept;

const float DRMAX = 20.*PI/180.;
const float rudderDCMAX = .102;
const float rudderDCMIN = .07;
float rudderDCslope;
float rudderDCintercept;

matrix massMatrix, invMassMatrix;
vector F, Fhat, S_depth, S_heading, PHIx, RUVstateVector, globalTempVec;
matrix G, Ghat, J, dPHIx;

const float xg = 0.001;
const float yg = -0.001;
const float zg = .0508;
const float xb = 0;
const float yb = 0;
const float zb = 0;

const float Ix = .0196;
const float Iy = 7.26;
const float Iz = 7.26;
const float Ixy = 0;
const float Ixz = -.02120;
```

```
const float Iyz = -.02120;


const float B = 124.73;
const float W = 124.727;
const float g = 9.81;
const float m = 12.7;         //W/g

const float Nrdot = -2.918;
const float Xudot = -.212;
const float Yvdot = -19.63;
const float Zwdot = -19.63;
const float Kpdot = -.0627;
const float Mqdot = -2.918;
const float Mwdot = 2.03;
const float Yrdot =  -2.03;
const float Nvdot =  -2.03;
const float Zqdot = 2.03;

const float Yur =  4.455;
const float Yuv = -22.4;
const float Zuq = -4.455;
const float Zuw = -22.4;
const float Muq = -4.52;
const float Muw =  11.85;
const float Nuv = -11.85;;

const float Xuu = -3.1419;
const float Yvv = -445.32;
const float Yrr =  2.86;
const float Zww = -445.32;
const float Zqq = -2.86;
const float Kpp = -2.6;
const float Mww =  2.95;
const float Mqq = -34.475;
const float Nvv = -2.95;
const float Nrr = -27.81;  //structured uncertainty

const float Yuudr = 8.75;
const float Zuuds = -8.75;
const float Muuds = -4.67;
const float Nuudr = -4.67;

void initRUVModel(void)
{

        massMatrix = mat(numStates,numStates,"");
        invMassMatrix = mat(numStates,numStates,"");
        F = vec(numStates,"");                          //temporary vector
        globalTempVec = vec(totStates,"");
        G = mat(numStates,numInputs,"");                //temporary matrix
        Ghat  = mat(totStates,numInputs,"");            //holds the input matrix estimate
        J = mat(numStates,numStates,"");                //Euler rotation matrix
        Fhat = vec(totStates,"");                       //holds the state deriv i.e. xdot
estimate
        RUVstateVector = vec(totStates,"");

        //Compute stern angle to duty cycle slope and intercept
        sternDCslope = (sternDCMAX - sternDCMIN)/(-2*DSMAX);
        sternDCintercept = (sternDCMAX + sternDCMIN)/2;

        //Compute rudder angle to duty cycle slope and intercept
        rudderDCslope = (rudderDCMAX - rudderDCMIN)/(-2*DRMAX);
        rudderDCintercept = (rudderDCMAX + rudderDCMIN)/2;
```

```
        S_depth = vec(totStates,"");
        zerovec(S_depth,totStates);
        S_depth[1] =  -0.5; S_depth[2] =  3.0; S_depth[3] =  0.3; S_depth[4] =  1.0;

        S_heading = vec(totStates,"");
        zerovec(S_heading,totStates);
        S_heading[1] =  1.0; S_heading[2] =  1.0; S_heading[3] =  1.0;

        PHIx = vec(totStates,"");
        dPHIx = mat(totStates,totStates,"");

        zeromat(massMatrix,numStates,numStates);
        massMatrix[1][1] = m-Xudot;  massMatrix[1][5] = m*zg;  massMatrix[1][6] = -m*yg;
        massMatrix[2][2] = m-Yvdot;  massMatrix[2][4] = -m*zg;  massMatrix[2][6] = m*xg -
Yrdot;
        massMatrix[3][3] = m-Zwdot;  massMatrix[3][4] = m*yg;  massMatrix[3][5] = -m*xg -
Zqdot;
        massMatrix[4][2] = -m*zg;  massMatrix[4][3] = m*yg;  massMatrix[4][4] = Ix-
Kpdot;massMatrix[4][5] = -Ixy;  massMatrix[4][6] = -Ixz;
        massMatrix[5][1] = m*zg;  massMatrix[5][3] = -m*xg-Mwdot;  massMatrix[5][4] = -
Ixy; massMatrix[5][5] = Iy-Mqdot; massMatrix[5][6] = -Iyz;
        massMatrix[6][1] = -m*yg;  massMatrix[6][2] = m*xg-Nvdot;  massMatrix[6][4] = -
Ixz; massMatrix[6][5] = -Iyz; massMatrix[6][6] =Iz-Nrdot;
        //showmat("massMat",massMatrix, numStates, numStates,0);

        inv(massMatrix ,numStates,invMassMatrix );
        //showmat("massMat",invMassMatrix, numStates, numStates,0);
        //printf("invM[5][6] = %e",invMassMatrix[5][6]);pause("");
}


void calcRUVmodel(float t,vector x, float *alpha_h, float *beta_h, float *alpha_d, float
*beta_d)    //use this with actual sm controller implementation
{

  float u, v, w, p, q, r, X, Y, Z, phi, theta, psi;
  float XHS=0, YHS=0, ZHS=0, KHS=0, MHS=0, NHS=0;


  u = x[1]; v = x[2]; w = x[3]; p = x[4]; q = x[5]; r = x[6];
  //X = x[7]; Y = x[8]; Z = x[9];
  phi = x[10]; theta = x[11]; psi = x[12];

  // if theta > 2*pi
  //    theta=theta-2*pi;
  // else if  theta < 0


  XHS = -(W-B)*sin(theta);
  //printf("XHS = %e",XHS);pause("");
  YHS = (W-B)*cos(theta)*sin(phi);
  //printf("YHS = %e",YHS);pause("");
  ZHS = (W-B)*cos(theta)*cos(phi);
  //printf("ZHS = %e",ZHS);pause("");
  _fpreset();
  KHS = (yg*W-yb*B)*cos(theta)*cos(phi)-(zg*W-zb*B)*cos(theta)*sin(phi);
  //printf("KHS = %e",KHS);pause("");
  _fpreset();
  MHS = -(zg*W-zb*B)*sin(theta)-(xg*W-xb*B)*cos(theta)*cos(phi);
  //printf("MHS = %e",MHS);pause("");
  _fpreset();
  NHS = (xg*W-xb*B)*cos(theta)*sin(phi)-(yg*W-yb*B)*sin(theta);
  //printf("NHS = %e",NHS);pause("");

  ////////////////////////////////////////////////////////////////
```

```
        zeromat(G, numStates, numInputs);
        G[1][3] = 1;
        G[2][1] = Yuudr*u*u;
        G[3][2] = Zuuds*u*u;
        G[4][4] = 1;
        G[5][2] = Muuds*u*u;
        G[6][1] = Nuudr*u*u;
        matmat(invMassMatrix, G, numStates, numStates, numInputs, G);
        //showmat("Ghat",Ghat, numStates,numInputs,0);
        Ghat[1][1] = G[1][1];Ghat[1][2] = G[1][2];Ghat[1][3] = G[1][3];Ghat[1][4] =
G[1][4];Ghat[1][5] = G[1][5];Ghat[1][6] = G[1][6];
        Ghat[2][1] = G[2][1];Ghat[2][2] = G[2][2];Ghat[2][3] = G[2][3];Ghat[2][4] =
G[2][4];Ghat[2][5] = G[2][5];Ghat[2][6] = G[2][6];
        Ghat[3][1] = G[3][1];Ghat[3][2] = G[3][2];Ghat[3][3] = G[3][3];Ghat[3][4] =
G[3][4];Ghat[3][5] = G[3][5];Ghat[3][6] = G[3][6];
        Ghat[4][1] = G[4][1];Ghat[4][2] = G[4][2];Ghat[4][3] = G[4][3];Ghat[4][4] =
G[4][4];Ghat[4][5] = G[4][5];Ghat[4][6] = G[4][6];
        Ghat[5][1] = G[5][1];Ghat[5][2] = G[5][2];Ghat[5][3] = G[5][3];Ghat[5][4] =
G[5][4];Ghat[5][5] = G[5][5];Ghat[5][6] = G[5][6];
        Ghat[6][1] = G[6][1];Ghat[6][2] = G[6][2];Ghat[6][3] = G[6][3];Ghat[6][4] =
G[6][4];Ghat[6][5] = G[6][5];Ghat[6][6] = G[6][6];
        //showmat("Ghat",Ghat, numStates,numInputs,0);

        //zerovec(F, numStates);
        F[1]=-m*(w*q-v*r-
xg*(r*r+q*q)+yg*q*p+zg*r*p)+XHS+Xuu*u*fabs(u)+Zwdot*w*q+Zqdot*q*q+Yvdot*v*r+Yrdot*r*r;
        F[2]=-m*(u*r-p*w+xg*p*q-yg*(r*r+p*p)+zg*r*q)+YHS+Yvv*v*fabs(v)+Yrr*r*fabs(r)-
Zwdot*w*p-Zqdot*q*p+Yur*u*r+Yuv*u*v;
        F[3]=-m*(v*p-q*u+xg*r*p+yg*r*q-
zg*(q*q+p*p))+ZHS+Zww*w*fabs(w);//+Zqq*q*fabs(q)+Yvdot*v*p+Yrdot*r*p+Zuq*q*u+Zuw*w*u;
        F[4]=-Ixy*p*r+Ixz*p*q-Ixz*(r*r-q*q)-(Iz-Iy)*r*q-m*(yg*(v*p-u*q)-zg*(u*r-
w*p))+KHS+Kpp*p*fabs(p)+((Zwdot-Yvdot)*w*v+(Zqdot+Nvdot)*v*q-(Yrdot+Mwdot)*r*w+(Nrdot-
Mqdot)*r*q);
        F[5]=Ixy*q*r-Iyz*p*q+Ixz*(r*r-p*p)-(Ix-Iz)*p*r-m*(zg*(w*q-v*r)-xg*(v*p-
u*q))+MHS+Mww*w*fabs(w)+Mqq*q*fabs(q)+Muq*u*q-Nvdot*v*p+Muw*u*w+(Kpdot-Nrdot)*r*p ;
        F[6]=Ixz*q*r+Iyz*r*p+Ixy*(p*p+q*q)+(Ix-Iy)*p*q-m*(xg*(u*r-w*p)-yg*(w*q-
v*r))+NHS+Nvv*v*fabs(v)+Nrr*r*fabs(r)+Nuv*u*v+Yrdot*u*r+Mwdot*w*p+(Mqdot-Kpdot)*p*q;
        //printf("Zww*w*fabs(w) = %e",Zww*w*fabs(w));pause("");
        //showvec("x",x,totStates,0);
        //showvec("F",F,numStates,0);

        //zeromat(J, numStates, numStates);
        J[1][1]=cos(psi)*cos(theta) ; J[1][2]= -
sin(psi)*cos(phi)+cos(psi)*sin(theta)*sin(psi); J[1][3]=
sin(psi)*sin(phi)+cos(psi)*cos(phi)*sin(theta); J[1][4]=   0; J[1][5]=  0; J[1][6]=  0;
        J[2][1]= sin(psi)*cos(theta); J[2][2]=
cos(psi)*cos(phi)+sin(phi)*sin(theta)*sin(psi); J[2][3]=  -
cos(psi)*sin(phi)+sin(theta)*sin(psi)*cos(phi); J[2][4]=  0; J[2][5]=  0; J[2][6]=  0;
        J[3][1]=  -sin(theta); J[3][2]= cos(theta)*sin(phi); J[3][3] = cos(theta)*cos(phi)
; J[3][4]=  0; J[3][5]=  0; J[3][6]=  0;
        J[4][1]= 0; J[4][2]=  0; J[4][3]=  0; J[4][4]=  1; J[4][5]=  sin(phi)*tan(theta);
J[4][6] = cos(phi)*tan(theta);
        J[5][1]= 0; J[5][2]=  0; J[5][3]=  0; J[5][4]=  0; J[5][5]=    cos(phi); J[5][6]=
-sin(phi);
        J[6][1]= 0; J[6][2]=  0; J[6][3]=  0; J[6][4]=  0; J[6][5]=
sin(phi)/cos(theta); J[6][6]=  cos(phi)/cos(theta);
        //showmat("J",J, numStates,numStates,0);

         matvec(invMassMatrix, F, numStates, numStates, F);
        Fhat[1] = F[1] ;Fhat[2] = F[2] ;Fhat[3] = F[3] ;Fhat[4] = F[4];Fhat[5] =
F[5];Fhat[6] = F[6];
        //showvec("f1",F,numStates,0);
        //zerovec(tempVec1, numStates);
        F[1] = u; F[2] = v; F[3] = w; F[4] = p; F[5] = q; F[6] = r;
        matvec(J, F, numStates, numStates, F ); // F = J*[u v w p q r]'
```

```
	Fhat[7] = F[1] ; Fhat[8] =  F[2];Fhat[9] =  F[3]; Fhat[10] =  F[4];  Fhat[11] =
F[5]; Fhat[12] = F[6]  ;
	//showvec("Fhat",Fhat,numStates,0);

	*alpha_h = Yuudr*u*u/(m-Yvdot);
	*beta_h = Nuudr*u*u/(Iz - Nrdot);
	*alpha_d = Zuuds*u*u/(m-Zwdot);
	*beta_d = Muuds*u*u/(Iy-Mqdot);



}

void calcRegCoordTrans(float t, vector x, vector xd, float a, float b, int
whichTransform)
{
	float u, v, w, p, q, r, X, Y, Z, phi, theta, psi;
	float ud, vd, wd, pd, qd, rd, Xd, Yd, Zd, phid, thetad, psid;

	u = x[1]; v = x[2]; w = x[3]; p = x[4]; q = x[5]; r = x[6]; phi = x[10]; theta =
x[11]; psi = x[12];
	X = x[7]; Y = x[8]; Z = x[9];
	ud = xd[1]; vd = xd[2]; wd = xd[3]; pd = xd[4]; qd = xd[5]; rd = xd[6]; phid =
xd[10]; thetad = xd[11]; psid = xd[12];
	Xd = xd[7]; Yd = xd[8]; Zd = xd[9];

	zeromat(dPHIx,totStates,totStates);

	if (whichTransform)  //do the depth transform
	{
		dPHIx[1][9] = 1;
		dPHIx[2][11] = 1;
		dPHIx[3][3] = b; dPHIx[3][5] = -a;
		dPHIx[4][5] = 1;
		dPHIx[5][1] = 1;
		dPHIx[6][2] =1;
		dPHIx[7][4] =1;
		dPHIx[8][6] =1;
		dPHIx[9][7] =1;
		dPHIx[10][8] =1;
		dPHIx[11][10] =1;
		dPHIx[12][12] =1;
		//showvec("Xd",xd,totStates,0);
		//showvec("X",x,totStates,0);
		PHIx[1] = Z - Zd;
		PHIx[2] = theta - thetad;
		PHIx[3] = b*(w-wd)-a*(q-qd);
		PHIx[4] = q-qd;
		PHIx[5] = u-ud;
		PHIx[6] = v-vd;
		PHIx[7] = p-pd;
		PHIx[8] = r-rd;
		PHIx[9] = X-Xd;
		PHIx[10] = Y-Yd;
		PHIx[11] = phi-phid;
		PHIx[12] = psi-psid;
	}
	else
	{
		dPHIx[1][9] = 1;
		dPHIx[2][2] = b; dPHIx[2][6] = -a;
		dPHIx[3][6] = 1;
		dPHIx[4][1] = 1;
		dPHIx[5][3] = 1;
		dPHIx[6][4] =1;
		dPHIx[7][5] =1;
```

```
                dPHIx[8][7] =1;
                dPHIx[9][8] =1;
                dPHIx[10][9] =1;
                dPHIx[11][10] =1;
                dPHIx[12][11] =1;

                PHIx[1] = psi-psid;
                PHIx[2] = b*(v-vd)-a*(r-rd);
                PHIx[3] = r-rd;
                PHIx[4] = u-ud;
                PHIx[5] = w-wd;
                PHIx[6] = p-pd;
                PHIx[7] =  q-qd;
                PHIx[8] = X-Xd;
                PHIx[9] = Y-Yd;
                PHIx[10] = Z-Zd;
                PHIx[11] = phi-phid;
                PHIx[12] = theta - thetad;
        }

}
```

Auxiliary Routines Code: ADAPT11.C

The routines in this section support reading the sensors on the DOS Stamp, communicating to the Adapt11 microcontroller, actuating the stern and rudder fins, and calculating the PID and sliding mode autopilot commands.

```
#include <dos.h>
#include <stdio.h>
#include <string.h>
#include <float.h>
#include <math.h>



#include "c:\borlandc\bin\phd\dosstamp\stdinc.h"
#include "c:\borlandc\bin\phd\dosstamp\dosstamp.h"
#include "c:\borlandc\bin\phd\dosstamp\dsrtc.h"
#include "c:\borlandc\bin\phd\dosstamp\dsadc.h"
#include "c:\borlandc\bin\phd\dosstamp\tdscomm.h"
#include "c:\borlandc\bin\phd\dosstamp\adapt11.h"

UBYTE PORT_MOTOR = 0;
UBYTE STBD_MOTOR = 1;

//some PID globals
float Ui_head=0;
float Ehead_=0;
float Ui_theta=0;
float Etheta_=0;
```

```
float deltaT = .33;




void setMotorDutyCycle(UBYTE motor, float dutyCycle, UWORD PERIOD_CYCLES){//port(0)mot
=portA5,starboard(1)mot=portA6
        UWORD hi, lo;
        static char *dutyCycle_A5_MemMap[2]={"0018","001a"};
        static char *dutyCycle_A6_MemMap[2]={"0020","0022"};
        char tempTxArrHi[10], tempTxArrLo[10], tempRxArr[4];
        char copyArr[4], copyArr1[4];

        hi = (UWORD)(dutyCycle*PERIOD_CYCLES);
        lo  = (PERIOD_CYCLES - hi);

        strcpy(copyArr1,"0");
        strcpy(copyArr,"0");                                    //initialize string arrays
        strcpy(tempTxArrHi,"0");
        strcpy(tempTxArrLo,"0");

        int2x(copyArr1, lo);
        zeropad(copyArr1,4);

        strcpy(tempTxArrLo, "z");
        strcat(tempTxArrLo, copyArr1);

        if(motor)
                strcat(tempTxArrLo, dutyCycle_A6_MemMap[1]);
        else
                strcat(tempTxArrLo, dutyCycle_A5_MemMap[1]);
        //printf("\nLow motor word: %s", tempTxArrLo);

        int2x(copyArr, hi);
        zeropad(copyArr,4);

        strcpy(tempTxArrHi,"z");
        strcat(tempTxArrHi, copyArr);

        if(motor) // pwm starboard motor
                strcat(tempTxArrHi,dutyCycle_A6_MemMap[0]);
        else
                strcat(tempTxArrHi,dutyCycle_A5_MemMap[0]);
        //printf("\nHigh motor word: %s", tempTxArrHi);
        SERreadwrite(tempTxArrHi,tempRxArr,6,TRUE); //send motor word to adapt11
        SERreadwrite(tempTxArrLo,tempRxArr,6,TRUE); //send motor word to adapt11


}

void setServoDutyCycle(float dutyCycle, UWORD PERIOD_CYCLES){ // servo motor connected to
6811's pa3
        UWORD hi, lo;
        static char *dutyCycle_A3_MemMap[2]={"001c","001e"};
        char tempTxArrHi[10], tempTxArrLo[10], tempRxArr[4];
        char copyArr[4], copyArr1[4];

        //printf("%.3f\n",dutyCycle);
        hi = (UWORD)(dutyCycle*PERIOD_CYCLES);
        //printf("%u\n", hi);
        lo  = (PERIOD_CYCLES - hi);
        //printf("%u\n", lo);

        strcpy(copyArr1,"0");
```

```
        strcpy(copyArr,"0");                                    //initialize string arrays
        strcpy(tempTxArrHi,"0");
        strcpy(tempTxArrLo,"0");

        int2x(copyArr1, lo);
        zeropad(copyArr1,4);

        strcpy(tempTxArrLo, "z");
        strcat(tempTxArrLo, copyArr1);


        strcat(tempTxArrLo, dutyCycle_A3_MemMap[1]);

        //printf("\nLow motor word: %s", tempTxArrLo);

        int2x(copyArr, hi);
        zeropad(copyArr,4);

        strcpy(tempTxArrHi,"z");
        strcat(tempTxArrHi, copyArr);


        strcat(tempTxArrHi,dutyCycle_A3_MemMap[0]);


        //printf("\nHigh motor word: %s", tempTxArrHi);
        SERreadwrite(tempTxArrHi,tempRxArr,6,TRUE); //send motor word to adapt11
        SERreadwrite(tempTxArrLo,tempRxArr,6,TRUE); //send motor word to adapt11


}


void setMotorONLine(UBYTE mot){
        char tempTxArr[10],  tempRxArr[4], *motorLineMemMap[2] = {"0024","0025"};

        strcpy(tempTxArr,"w00ff");

        if(mot)
                strcat(tempTxArr,motorLineMemMap[1]);
        else
                strcat(tempTxArr,motorLineMemMap[0]);

        SERreadwrite(tempTxArr, tempRxArr,4,TRUE);
}

void setMotorOFFLine(UBYTE mot){
        char tempTxArr[10],  tempRxArr[4], *motorLineMemMap[2] = {"0024","0025"};

        strcpy(tempTxArr,"w0000");

        if(mot)
                strcat(tempTxArr,motorLineMemMap[1]);
        else
                strcat(tempTxArr,motorLineMemMap[0]);

        SERreadwrite(tempTxArr, tempRxArr,4,TRUE);
}

void setMotorLine(UBYTE mot, UBYTE line){

        if(line) setMotorONLine(mot);
        else     setMotorOFFLine(mot);

}
```

```
void setBothMotorsONLine(void){

        setMotorONLine(STBD_MOTOR);
        setMotorONLine(PORT_MOTOR);
}

void setBothMotorsOFFLine(void){

        setMotorOFFLine(STBD_MOTOR);
        setMotorOFFLine(PORT_MOTOR);

}

void setMotorDirection(UBYTE mot, UBYTE dir){                        //writes   to   6811's
port c to change direction of desired motor
        char tempTxArr[10],  tempRxArr[4],  copyArr[1];
        UBYTE  digVal = 0;


        strcpy(copyArr,"0");
        strcpy(tempRxArr, "0");


        strcpy(tempTxArr,"r00001003");
        SERreadwrite(tempTxArr,tempRxArr,4,FALSE);
        getxsubstr(tempRxArr);
        //strcpy(tempRxArr,"ff");
        digVal = (UBYTE)x2int(tempRxArr);

        if(mot)
                if(dir)                //turn stbd motor one dir (wrt 1 to c7)
                        digVal |= 0x80;
                else                            //shut  stbd  motor  the  other  dir  (wrt  0  to
c7)
                        digVal &= 0x7F;
        else
                if(dir)
                        digVal  |= 0x40;              //turn  port  motor  one  dir  (wrt  1  to
c6)
                else
                        digVal  &= 0xBF;              //turn  port  motor  the  other  dir  (wrt
0 to c6)

        int2x(copyArr, (UWORD)digVal);
        zeropad(copyArr,4);
        strcpy(tempTxArr, "w");
        strcat(tempTxArr, copyArr);
        strcat(tempTxArr, "1003");

        //printf("\nsetMotorLine outputs: %s", tempTxArr);
        SERreadwrite(tempTxArr, tempRxArr, 4, TRUE);    //send string "wxxxx1003"
}

void setBothMotorsReverse(void){
        setMotorDirection(PORT_MOTOR,0);
        setMotorDirection(STBD_MOTOR,0);
}

void setBothMotorsForward(void){
        setMotorDirection(PORT_MOTOR,1);
        setMotorDirection(STBD_MOTOR,1);
}

UBYTE getAnalogPort(UBYTE anport){
        static char *anportMemMap[8]={"04", "05", "06", "07", "08", "09","0a","0b"};
        char tempTxArr[10];
```

```
        char tempRxArr[4];


        strcpy(tempTxArr,"r000000");
        strcat(tempTxArr,anportMemMap[anport]);
        SERreadwrite(tempTxArr,tempRxArr,4,FALSE);
        getxsubstr(tempRxArr);
        return (UBYTE)x2int(tempRxArr);

}

UBYTE getDigitalPort(UBYTE digport){
        char tempTxArr[10];
        char tempRxArr[4];
        UBYTE  digArr[8] = {1,2,4,8,16,32,64,128};

        digport = digArr[digport];
        strcpy(tempTxArr,"r00001003");
        SERreadwrite(tempTxArr,tempRxArr,4,FALSE);
        getxsubstr(tempRxArr);

        return !!(digport&~(UBYTE)x2int(tempRxArr));
}

void adapt11Reset(void){
        int i=0;

        PIOconfig(11,PIOCFG_OUTPUT,0);
        for(i = 0; i < 1000; i++);           //induce  a  slight  delay  before  going  high
again
        PIOconfig(11,PIOCFG_OUTPUT,1);
}

float getDepthSensor(void){   //depth sensor connected to dos stamps adc input 2
        int numAvg = 100;
        WORD chan=2;
        float v =0;

        v = (.0012207)*ADCgetAvgNSamples(chan, ADC_0_TO_5, numAvg);

        return 25*(v-.512)*.7028;//depth in meters ->  1psi = 27.67" * ( 1m/ 39.37")
}
float getBatteryLevel(void){
        int numAvg = 10;
        WORD chan=0;
        float v =0;

        v = 4.72*(.0012207)*ADCgetAvgNSamples(chan, ADC_0_TO_5, numAvg);
        return v;  //battery level in volts
}

float getPitchSensor(void){
        int numAvg = 10;
        WORD chan=6;
        float v =0, v_0  = 2.539, sensitivity = 34.882, theta;

        v = (.0012207)* ADCgetAvgNSamples(chan, ADC_0_TO_5, numAvg);
        v=(v - v_0)*.5004;
        if (v>1.5)
                theta = 90;
        else if (v <-1.5)
                theta = -90;
        else
                theta =57.2958*asin(v);  //.5029 = 1000/sensitivity/57.2958
        return (theta-(-1.2697));  //pitch angle in degrees
```

```
}
float getPitchRateSensor(){  //returns pitch rate in degrees/sec
//      int i=0,
        int numAvg = 10;
        WORD chan=5;

        //return   ((300.0/4.5)*(.0012207)*ADCgetAvgNSamples(chan,   ADC_0_TO_5,   numAvg)-
148.2);
        return   -(300.0/4.5)*((.0012207)*ADCgetAvgNSamples(chan,   ADC_0_TO_5,   numAvg)-
2.2278);
}
float getYawRateSensor(){     //returns yaw rate in degrees/sec
        int numAvg = 10;
        WORD chan=7;

        //return   ((300.0/4.5)*(.0012207)*ADCgetAvgNSamples(chan,   ADC_0_TO_5,   numAvg)-
170.4);
        return    (300.0/4.5)*((.0012207)*ADCgetAvgNSamples(chan,    ADC_0_TO_5,    numAvg)-
2.5565);
}
float getRollSensor(void){
        int numAvg = 10;
        WORD chan=4;

        return (.0012207)*ADCgetAvgNSamples(chan, ADC_0_TO_5, numAvg);
}

int getCompassReading(void){  //returns heading in degrees
        char tempTxArr[10];
        char tempRxArr[6];

        strcpy(tempTxArr,"y00000000");
        SERreadwrite(tempTxArr,tempRxArr,6,FALSE);

        getxsubstr(tempRxArr);
        //printf(tempRxArr);
        //printf("\n");                      //strip off send chars
        return (360-(UWORD)x2int(tempRxArr));
}



void setLED(UBYTE LED, UBYTE state){                //writes to 6811's port c to change
led status
        char tempTxArr[10],  tempRxArr[4],  copyArr[1];
        UBYTE  digVal = 0;


        strcpy(copyArr,"0");
        strcpy(tempRxArr, "0");


        strcpy(tempTxArr,"r00001003");
        SERreadwrite(tempTxArr,tempRxArr,4,FALSE);
        getxsubstr(tempRxArr);
        //strcpy(tempRxArr,"ff");
        digVal = (UBYTE)x2int(tempRxArr);

        if(!LED)
                if(state)                                       //red led state
                        digVal |= 0x01;
                else
                        digVal &= 0xFE;
        else
                if(state)                                       // green led state
                        digVal  |= 0x02;              // (wrt 1 to c1)
```

```
                else
                        digVal  &= 0xFD;                // (wrt 0 to c1)

        int2x(copyArr, (UWORD)digVal);
        zeropad(copyArr,4);
        strcpy(tempTxArr, "w");
        strcat(tempTxArr, copyArr);
        strcat(tempTxArr, "1003");

        SERreadwrite(tempTxArr, tempRxArr, 4, TRUE);    //send string "wxxxx1003"
}
void getRUVstate(vector x)
{
        zerovec(x,totStates);
        //u v w p q r X Y Z phi theta psi
        x[1] = 1.;                              // typical u in m/sec
        x[2] = 0;                               // v in m/sec
        x[3] = 0;                               // w in m/sec
        x[4] = 0;                               // p in rad/sec
        x[5] = getPitchRateSensor()/57.2958;            // q  in rad/sec
        x[6] = getYawRateSensor()/57.2958;              // r in rad/sec
        x[7] = 0;                               // X in meters  -    don't    require    for
autopilots
        x[8] = 0;                               //   Y   in   meters   -   don't   require   for
autopilots
        x[9] = getDepthSensor();                //Z in meters
        x[10] = 0;                              //  roll  angle,  phi  in  radians    -  don't
require (passively stable in roll)
        x[11] = getPitchSensor()/57.2958;       //pitch angle, theta in radians
        x[12]= (float)getCompassReading()/57.2958;   //heading, psi in radians


        /*the following state is for testing autoPilotSM/PID numerical accuracy against
matlab routines - cf controller_PID.m */
//      x[1]=1.445;x[2]=                                                        -0.06478;x[3]=-
0.06538;x[4]=0.05336;x[5]=0.09675;x[6]=0.08042;x[7]=2.765;
//      x[8]=29.16;;x[9]=7.004;x[10]=.06893;x[11]=-.5443;;x[12]= 3.119;
        fprintf(LOGFILENAME, "%.1f\t%.2f\t%.2f\t%.2f\t%.2f\t%.3f\t%.3f\n",
         x[12] ,x[6] ,x[11],x[5],x[9],RUDDER_DC,STERN_DC);
}
void autoPilotSM(float t, vector Xd_depth, vector Xd_heading, UWORD motPeriod)
{
        float alpha_h, beta_h, alpha_d, beta_d;
        float eta, delta, sigma, invVal, tempVal;
        float sternDutyCycle, rudderDutyCycle;
        float ds, dr;

        getRUVstate(RUVstateVector);   //updates RUVstateVector
        calcRUVmodel(t,RUVstateVector,  &alpha_h,  &beta_h,  &alpha_d,  &beta_d);  //updates
Fhat, Ghat, alpha, beta

        //calculate the stern command
        calcRegCoordTrans(t,RUVstateVector,  Xd_depth,  alpha_d,  beta_d,  1);   //calc  PHIx,
dPHIx
        eta = 10;
        delta = 1;
        getcol(Ghat,totStates,totStates,2, globalTempVec);
        matvec(dPHIx,globalTempVec,totStates,totStates,globalTempVec);
        invVal = inner(S_depth,globalTempVec,totStates);
        sigma = inner(S_depth,PHIx,totStates);
        matvec(dPHIx,Fhat,totStates,totStates,globalTempVec);
        tempVal = inner(S_depth,globalTempVec,totStates);
        ds = - (1./invVal)*tempVal - (1./invVal)*eta*tanh(sigma/delta);
        //shownum("ds",ds);
        if (ds > DSMAX)
                sternDutyCycle = sternDCMIN;
```

```
        else if (ds < -DSMAX)
                sternDutyCycle = sternDCMAX;
        else
                sternDutyCycle = sternDCslope*ds+sternDCintercept;
        //printf("sternDutyCycle = %f\n",sternDutyCycle);
        setMotorDutyCycle(1,sternDutyCycle, motPeriod);
        STERN_DC= sternDutyCycle;

        //calculate the rudder command
        calcRegCoordTrans(t,RUVstateVector, Xd_heading, alpha_h, beta_h, 0);  //calc PHIx,
dPHIx
        eta = 10;
        delta = 1;
        getcol(Ghat,totStates,totStates,1, globalTempVec);
        matvec(dPHIx,globalTempVec,totStates,totStates,globalTempVec);
        invVal = inner(S_heading,globalTempVec,totStates);
        sigma = inner(S_heading,PHIx,totStates);
        matvec(dPHIx,Fhat,totStates,totStates,globalTempVec);
        tempVal = inner(S_heading,globalTempVec,totStates);
        dr = - (1./invVal)*tempVal - (1./invVal)*eta*tanh(sigma/delta);
        //shownum("dr",dr);
        if (dr > DRMAX)
                rudderDutyCycle = rudderDCMIN;
        else if (dr < -DRMAX)
                rudderDutyCycle = rudderDCMAX;
        else
                rudderDutyCycle = rudderDCslope*dr + rudderDCintercept;
        //printf("rudderDutyCycle = %f\n",rudderDutyCycle);
        setServoDutyCycle(rudderDutyCycle, motPeriod);
         RUDDER_DC= rudderDutyCycle;
}

void autoPilotPID(float t, vector Xd_depth, vector Xd_heading, UWORD motPeriod)
{
        float sternDutyCycle, rudderDutyCycle;
        float Z, psi, theta, etheta, epsi, ez, thetad;
        float ds, dr;
        float Kp_z;
        float Kp_theta;
         float Ki_theta;
         float Kd_theta;

        float Kp_head;
         float Ki_head;
         float Kd_head;

        float derivTerm, propTerm, integralTerm;
        float maxIntegralTerm, minIntegralTerm;

        getRUVstate(RUVstateVector);   //updates RUVstateVector
        psi = RUVstateVector[12];     //heading in radians
        Z = RUVstateVector[9];        //depth
        theta = RUVstateVector[11];  //pitch
         //DEPTH
         Kp_z = 0.1667;
         ez= Z - Xd_depth[9];
         thetad=Kp_z*ez;

         Kp_theta = -110.8;
         Ki_theta = -1.17e4;
         Kd_theta = -2.72e5;


         maxIntegralTerm = 10;
         minIntegralTerm = -maxIntegralTerm;
```

```
        etheta = theta - thetad;//shownum("etheta",etheta);

        propTerm = Kp_theta*etheta;                    //proportional term
        integralTerm = Ki_theta*deltaT*etheta + Ui_theta;  //integral term
        if (integralTerm>maxIntegralTerm)
            integralTerm = maxIntegralTerm;
        if (integralTerm < minIntegralTerm)
            integralTerm = minIntegralTerm;

        Ui_theta = integralTerm;
        derivTerm = (Kd_theta/deltaT)*(etheta - Etheta_) ;     //derivative term
        Etheta_ = etheta;
        ds = -( 1.*propTerm+1.*integralTerm+1.*derivTerm);
        //printf("ds = %f\n",ds);
        if (ds > DSMAX)
               sternDutyCycle = sternDCMIN;
        else if (ds < -DSMAX)
               sternDutyCycle = sternDCMAX;
        else
               sternDutyCycle = sternDCslope*ds+sternDCintercept;
        //printf("sternDutyCycle = %f\n",sternDutyCycle);
        setMotorDutyCycle(1,sternDutyCycle, motPeriod);
        STERN_DC = sternDutyCycle;
        //HEADING:

        Kp_head = -54.5;
        Ki_head = -31.093;
        Kd_head = -11.049;


        maxIntegralTerm = 10;
        minIntegralTerm = -maxIntegralTerm;

        epsi = psi - Xd_heading[12];
        propTerm = Kp_head*epsi;                    //proportional term

        //integralTerm = Ki_head*deltaT*epsi + Ui_head;  //integral term
        //if (integralTerm>maxIntegralTerm)
        //    integralTerm = maxIntegralTerm;
        //if (integralTerm < minIntegralTerm)
        //    integralTerm = minIntegralTerm;
        //Ui_head = integralTerm;

        derivTerm = (Kd_head/deltaT)*(epsi - Ehead_);      //derivative term
        Ehead_ = epsi;
        //dr = -( 1.*propTerm+1.*integralTerm+1.*derivTerm);
        dr = -( 1.*propTerm+1.*derivTerm);
        //printf("dr = %f\n",dr);

        if (dr > DRMAX)
               rudderDutyCycle = rudderDCMIN;
        else if (dr < -DRMAX)
               rudderDutyCycle = rudderDCMAX;
        else
               rudderDutyCycle = rudderDCslope*dr + rudderDCintercept;
        //printf("rudderDutyCycle = %f\n",rudderDutyCycle);
       setServoDutyCycle(rudderDutyCycle, motPeriod);
       RUDDER_DC = rudderDutyCycle;
}
```

Adapt 11 Microcontroller Control Code

The Adapt 11 control code is an interrupt-driven assembly routine comprised of functions for reading the Adapt 11 analog-to-digital converters, SPI ports, digital I/O, and generating the PWM signals for the thruster and fin actuators. The hex-based serial communication protocol is based on assembly written for the Mini Board robot controller [Fred Martin].

```
* 6811 equates
EEPROM  EQU     $F800           ; start of eeprom

*****************************************************************
* Control Registers

BASE    EQU     $1000

PORTA   EQU     $1000   ; Port A data register
RESV1   EQU     $1001   ; Reserved
PIOC    EQU     $1002   ; Parallel I/O Control register
PORTC   EQU     $1003   ; Port C latched data register
PORTB   EQU     $1004   ; Port B data register
PORTCL  EQU     $1005   ;
DDRC    EQU     $1007   ; Data Direction register for port C
PORTD    EQU     $1008  ; Port D data register
DDRD     EQU     $1009  ; Data Direction register for port D
PORTE   EQU     $100A   ; Port E data register
CFORC   EQU     $100B   ; Timer Compare Force Register
OC1M    EQU     $100C   ; Output Compare 1 Mask register
OC1D    EQU     $100D   ; Output Compare 1 Data register

* Two-Byte Registers (High,Low -- Use Load & Store Double to access)
TCNT    EQU     $100E   ; Timer Count Register
TIC1    EQU     $1010   ; Timer Input Capture register 1
TIC2    EQU     $1012   ; Timer Input Capture register 2
TIC3    EQU     $1014   ; Timer Input Capture register 3
TOC1    EQU     $1016   ; Timer Output Compare register 1
TOC2    EQU     $1018   ; Timer Output Compare register 2
TOC3    EQU     $101A   ; Timer Output Compare register 3
TOC4    EQU     $101C   ; Timer Output Compare register 4
TI4O5   EQU     $101E   ; Timer Input compare 4 or Output compare 5 register

TCTL1   EQU     $1020   ; Timer Control register 1
TCTL2   EQU     $1021   ; Timer Control register 2
TMSK1   EQU     $1022   ; main Timer interrupt Mask register 1
TFLG1   EQU     $1023   ; main Timer interrupt Flag register 1
TMSK2   EQU     $1024   ; misc Timer interrupt Mask register 2
TFLG2   EQU     $1025   ; misc Timer interrupt Flag register 2
PACTL   EQU     $1026   ; Pulse Accumulator Control register
PACNT   EQU     $1027   ; Pulse Accumulator Count register
SPCR    EQU     $1028   ; SPI Control Register
SPSR    EQU     $1029   ; SPI Status Register
SPDR    EQU     $102A   ; SPI Data Register
BAUD    EQU     $102B   ; SCI Baud Rate Control Register
SCCR1   EQU     $102C   ; SCI Control Register 1
```

```
SCCR2   EQU     $102D   ; SCI Control Register 2
SCSR    EQU      $102E  ; SCI Status Register
SCDR    EQU      $102F  ; SCI Data Register
ADCTL   EQU     $1030   ; A/D Control/status Register
ADR1    EQU     $1031   ; A/D Result Register 1
ADR2    EQU     $1032   ; A/D Result Register 2
ADR3    EQU     $1033   ; A/D Result Register 3
ADR4    EQU     $1034   ; A/D Result Register 4
BPROT   EQU     $1035   ; Block Protect register
RESV2   EQU     $1036   ; Reserved
RESV3   EQU     $1037   ; Reserved
RESV4   EQU     $1038   ; Reserved
OPTION  EQU     $1039   ; system configuration Options
COPRST  EQU     $103A   ; Arm/Reset COP timer circuitry
PPROG   EQU     $103B   ; EEPROM Programming register
HPRIO   EQU     $103C   ; Highest Priority Interrupt and misc.
INIT    EQU     $103D   ; RAM and I/O Mapping Register
TEST1   EQU     $103E   ; factory Test register
CONFIG  EQU     $103F   ; Configuration Control Register


* Masks for serial port
PORTD_WOM       EQU     $20
BAUD1200        EQU     $B3
BAUD9600        EQU     $B0
TRENA           EQU     $0C     ; Transmit, Receive ENAble
RDRF            EQU     $20     ; Receive Data Register Full
TDRE            EQU     $80     ; Transmit Data Register Empty

* ASCII definitions
CR              EQU     $0a

* motor control
MOTORS_OFF      EQU     $00     ; rev 1 board

* stack location
STACK_LOC       EQU     $00FF
*****************************************************************
* zero page RAM definitions

        ORG     $00

st_hi   RMB     2
st_lo   RMB     2


* direct analog values (updated 1 kHz)
a0val   RMB     1
a1val   RMB     1
a2val   RMB     1
a3val   RMB     1
a4val   RMB     1
a5val   RMB     1
a6val   RMB     1
a7val   RMB     1


* readV2X result location (read using read q command)
heading RMB     2       * compass output location

* store calculated range time from the OAS
range_time      RMB     2               ; range value in tcnt cycles

* some flags used in the sonar routines
start_time      RMB     2               ; initial tcnt value to subtract from range_time
above
```

```
echo_capture   RMB    1              ; flag to tell system interrupt that IC3 interrupt
handler caught echo

*System Interrupt rate (default = 4kHz)
sysIntCyc      RMB    2              ; number of interrupt cycles

* Variable to store user specified value to flag range alarm
range_alarm    RMB    2
alarm_count    RMB    1

* variables used in pwm servo routine on PortA5
highA5         RMB    2
lowA5          RMB    2

* variables used in pwm servo routine on PortA3
highA3         RMB    2
lowA3          RMB    2

* variables used in pwm servo routine on PortA6
highA6         RMB    2
lowA6          RMB    2

*variables for turning on/off port and stbd motors connected to PA5 and PA6 resp.
portON         RMB    1
stbdON         RMB    1
*************************************************************************
*                                                                       *
*                            MAIN CODE                                  *
*                                                                       *
*************************************************************************

       ORG    $F800

Start:
soft_reset:
       LDS    #STACK_LOC

       LDX     #$1000

* pre-scale free-running counter
       LDAA   #$01           *  TMSK2=$01 => resolution=500cycles/msec
       STAA   TMSK2,X

* make pc6 and pc7 outputs for direction control of port & stbd motors, resp
       LDAA   #$C0
       STAA   DDRC


* it's a rev 1 board:  turn off the motors
       LDAA   #MOTORS_OFF
       STAA   PORTB,X

* initialize SPI port for V2X use
       LDX    #BASE
       LDAA   #%00111100     * ~SS (bit 5) is now general purpose output
       STAA   DDRD
       LDAA   #%01011111     * SPE = 1,master mode,CPOL=1,CPHA=1,1000khz
       STAA   SPCR,X
       LDAA   #$20           * raise ~SS line
       STAA   PORTD

* initialize serial port
       BCLR   SPCR,X PORTD_WOM      ; turn off wired-or mode
       LDAA   #BAUD9600
       STAA   BAUD,X
       LDAA   #TRENA
```

```
        STAA    SCCR2,X

* turn on analog subsystem
        BSET    OPTION,X $80

* set up interrupts
* system interrupt & IC3 OAS echo capture
        BSET    TFLG1,X %00010000      ; generate OC4 int immediately after init routine,
IC3 int started in range routine
        BSET    TMSK1,X %00010001      ; enable system interrupt (not IR) & IC3 interrupt

* setup for the pwm routine on OC3/PA5
        LDAA    TMSK1,X                ; old value
        ORAA    #$20                   ;TMSK1 OC3I = 1
        STAA    TMSK1,X                ; arm OC3 interrupt flag
        LDAA    TCTL1,X
        ORAA    #$30                   ; set OC3 output to one upon successful compare
        STAA    TCTL1,X
        LDAA    #$20
        STAA    TFLG1,X                ; clear OC3 flag
        LDD     #600                   ; start OC3 interrupts 1.2msec after init routine -
this allows system int
        STD     TOC3,X                 ; above to start 1st

* setup for the pwm routine on OC5/PA3
        LDAA    TMSK1,X                ; old value
        ORAA    #$08                   ;TMSK1 OC5I = 1
        STAA    TMSK1,X                ; arm OC5 interrupt flag
        LDAA    TCTL1,X
        ORAA    #$03                   ; set OC5 output to one upon successful compare
        STAA    TCTL1,X
        LDAA    #$08
        STAA    TFLG1,X                ; clear OC5 flag
        LDD     #700                   ; start OC5 interrupts 1.4msec after init routine -
this allows system int
        STD     TI4O5,X                ; to start 1st

* setup for the pwm routine on OC2/PA6
        LDAA    TMSK1,X                ; old value
        ORAA    #$40                   ;TMSK1 OC2I = 1
        STAA    TMSK1,X                ; arm OC2 interrupt flag
        LDAA    TCTL1,X
        ORAA    #$C0                   ; set OC2 output to one upon successful compare
        STAA    TCTL1,X
        LDAA    #$40
        STAA    TFLG1,X                ; clear OC2 flag
        LDD     #800                   ; start OC2 interrupts 1.6msec after init routine -
this allows system int
        STD     TOC2,X                 ; to start 1st


* initialize system time
        CLRA
        CLRB
        STD     st_hi
        STD     st_lo


        LDAA    #$1
        STAA    echo_capture   ; nonzero = no capture / zero = capture
        LDD     #$0
        STD     range_time     ; init range time to zero
        STD     start_time     ; init start time to zero

        LDD     #500
```

```
        STD     sysIntCyc       ; 4 kHz system interrupt rate (range divisor in nav_loop
must match)


        LDD     #30
        STD     range_alarm     ; trigger collision alarm at 30 inches

        LDD     #7780
        STD     lowA5
        LDD     #720
        STD     highA5          ; initialize pwm for high = 1.44 msec, period = 17.0 ms

        LDD     #7780
        STD     lowA3
        LDD     #720
        STD     highA3          ; initialize pwm for high = 1.44 msec

        LDD     #7780
        STD     lowA6
        LDD     #720
        STD     highA6          ; initialize pwm for high = 1.44 msec

        LDAA    #0
        STAA    portON          ; stbd motor initialized to off state

        LDAA    #0
        STAA    stbdON

* turn on interrupts
        CLI
        JMP     cmd_loop




*****************************************************************************************
*********************
*       COMMAND LOOP: commands used by dos stamp to maintain executive control
*                      a digit is a hex char (upcase only)
*                     x means ignored but required
*                read byte
*                      r addr: x x x x add3 add2 add1 add0 -> r dig dig
*                write byte
*                      w addr: x x dat1 dat0 add3 add2 add1 add0 -> w
*                read word
*                      q addr: x x x x dig dig dig dig -> q dig dig dig dig
*               write word
*                     z addr: dat3 dat2 dat1 dat0 add3 add2 add1 add0 -> z
*               jump to 6811 subroutine
*                     j : pchi pchi pclo pclo xreg xreg xreg xreg -> j
*                reset
*                      s : x x x x x x x x -> s
*
*       if read get address: print @address: goto cmd_loop
*       if write get address,data: poke address,data: goto cmd_loop
*       if call get address: push 0 on pstack: pc=address: jsr execute:
*              print return value (srhi): goto cmd_loop
*       if continue then rts
*       if reset goto reset
*
```

```
cmd_loop:
        LDAA    #'>
        JSR     putchar
* get a cmd character
* a command is always 'a' or bigger
* and data is always smaller than 'a' (@-O)
get_cmd_type:
        JSR     getchar
        CMPA    #'a
        BLO     get_cmd_type    * rcvd char was not a command character go back
        PSHA                    * save command type
* get 2 words of data
        JSR     getword         * data (lsb), new pc for call: y
        PSHX
        PULY
        JSR     getword         * address: x
* branch on cmd type
        PULA                    * restore command type
        jsr     putchar         * echo back command type
        CMPA    #'r
        BEQ     cmd_read_byte
        CMPA    #'w
        BEQ     cmd_write_byte
        CMPA    #'q
        BEQ     cmd_read_word
        CMPA    #'z
        BEQ     cmd_write_word
        CMPA    #'x
        BNE     cmd_loop_nl1
        JMP     cmd_initV2X
cmd_loop_nl1
        CMPA    #'y
        BNE     cmd_loop_nl2
        JMP     cmd_readV2X
cmd_loop_nl2
        CMPA    #'t
        BNE     cmd_loop_nl3
        JMP     cmd_range
cmd_loop_nl3
        CMPA    #'j
        BEQ     cmd_jsr
        BRA     cmd_loop


cmd_jsr
* jsr to y
        XGDY    * b now has y:lo, a is y:hi
        LDY     #cmd_loop   * save return address
        PSHY
        PSHB    * push y:lo as low of jump address
        PSHA    * push y:hi as hi of jump address
        RTS     * jump!

cmd_read_byte
        LDAA    0,X
        JSR     putbyte
        JMP     cmd_loop

cmd_read_word
        LDAA    0,X
        JSR     putbyte
        LDAA    1,X
        JSR     putbyte
        JMP     cmd_loop

cmd_write_byte
```

```
        XGDY    * move y to d.  low byte to poke is now in b
        STAB    0,X
        JMP     cmd_loop

cmd_write_word
        XGDY    * move y to d.
        STD     0,X
        JMP     cmd_loop

cmd_range
        LDX     #BASE
        LDD     #$0100
        SEI                             ; make atomic tx portion of this routine!
        LDY     TCNT,X                  * store t_0
        STY     start_time
*
        STAA    PORTC                   * genr8 10 cycles of 250khz
        STAB    PORTC                   * assumes accd=$0100
        STAA    PORTC
        STAB    PORTC
        STAA    PORTC
        STAB    PORTC
        STAA    PORTC
        STAB    PORTC
        STAA    PORTC
        STAB    PORTC
        STAA    PORTC
        STAB    PORTC
        STAA    PORTC
        STAB    PORTC
        STAA    PORTC
        STAB    PORTC
        STAA    PORTC
        STAB    PORTC
        STAA    PORTC
        STAB    PORTC
        STAA    PORTC
        STAB    PORTC
*
        LDAB    #$4C                    * wait 380 cylces (=76loops=190us=5.6") to clear
ringing
decloop1
        DECB                            * accounts for PA0 arming below
        BNE     decloop1
*
        LDAA    #$01                    ; enable rising edge detect on IC3 pin (disabled by
handler
        STAA    TCTL2,X                 ;       or if timeout, disabled by time_out code)
        CLI
*
        LDY     #$014C
decloop2
        DEY                             ; check for timeout after approx 4msec range (332
loops)
        BNE     decloop2                ; just loop while IC3 interrupt handler waits for
capture
*
        LDAA    echo_capture            ; check for echo capture
        CMPA    #$00
        BNE     time_out
*
        LDD     range_time              ; echo detected
        SUBD    start_time
        STD     range_time              ; send cycle count to serial
        COM     echo_capture            ; reset echo capture flag
        LDX     #range_time
        JMP     cmd_read_word
*
```

```
time_out
        LDD     #$0000
        STAA    TCTL2,X                 ; disable capture after 4msec range
        STD     range_time
        LDX     #range_time             ; send timeout signal (= 0) to serial
        JMP     cmd_read_word

cmd_initV2X
        LDX     #BASE
        LDAA    #%00111100      * ~SS (bit 5) is now general purpose output
        STAA    DDRD
        LDAA    #%01011111      * SPE = 1,master mode,CPOL=1,CPHA=1,1MHZ
        STAA    SPCR,X
        LDAA    #$20            * raise ~SS line
        STAA    PORTD
        JMP     cmd_loop

cmd_readV2X
        LDX     #BASE
line1
*       LDAA    #$00

        BCLR    PORTD,X %00100000
*       STAA    PORTD                   * drop ~P/C line to V2X (~SS)
        JSR     sleep10                 * then wait 10 msec
*       LDAA    #$20
*       STAA    PORTD                   * raise the ~P/C line
        BSET    PORTD,X %00100000
        JSR     sleep90                 * wait 90 msec
*       LDAA    #$00
*       STAA    PORTD                   * drop ~SS line
        BCLR    PORTD,X %00100000
        JSR     sleep10                 * wait 10 msec
        STAA    SPDR,X                  * start the SPI transfer
wait1
        BRCLR   SPSR,X $80 wait1        * wait for transfer of hi byte
        LDAA    SPDR,X                  * read the 1st byte - clears SPIF
        PSHA
        JSR     sleep10                 * wait 10 msec
        STAA    SPDR,X                  * start 2nd transfer
wait2
        BRCLR   SPSR,X $80 wait2        * wait for lo byte
        LDAB    SPDR,X                  * clear SPIF

*       LDAA    #$20
*       STAA    PORTD                   * raise ~SS line before exiting
        BSET    PORTD,X %00100000
        PULA
*       CPD     #$FFFF
*       BEQ     line1
        STD     heading
        LDX     #heading
        JMP     cmd_read_word

sleep1
        LDX     #BASE
        LDD     TCNT,X
        ADDD    #$1F4
delay1
        CPD     TCNT,X
        BGT     delay1
        RTS
*
sleep10
        LDX     #BASE
        LDD     TCNT,X
```

```
        ADDD    #$1388
delay10
        CPD     TCNT,X
        BGT     delay10
        RTS
*
sleep90
        LDX     #BASE
        LDD     TCNT,X
        ADDD    #$AFC8
delay90
        CPD     TCNT,X
        BGT     delay90
        RTS


* get a byte into A.  munges B
getbyte
        JSR     getchar
         CMPA    #'A
         BLO     hibyteok
         SUBA    #'A-10
hibyteok
        ASLA
        ASLA
        ASLA
        ASLA
        TAB
        JSR     getchar
         CMPA    #'A
         BLO     lobyteok
         SUBA    #'A-10
lobyteok
        ANDA    #$0f
        ABA
        RTS

* get a word into X
getword
        JSR     getbyte  * get hi byte first
        PSHA
        JSR     getbyte
        TAB
        PULA
        XGDX
        RTS

* get a character into A
getchar
        LDAA    SCSR
        ANDA    #RDRF
         BEQ    getchar
        LDAA    SCDR
         ANDA    #$7f
        RTS

* put a character from A.  munges B
putchar
        LDAB    SCSR
        ANDB    #TDRE
        BEQ     putchar
        STAA    SCDR
        RTS

* Prints out byte in A in Hex
putbyte
```

```
        JSR     Hex2Ascii
         PSHB
        JSR     putchar
         PULB
        TBA
        JMP     putchar


*******************************************************************************
*
*       Hex2Ascii:      converts byte to its 2-char ASCII hex equiv.
*
*                       INPUT:  byte in A register
*                       OUTPUT: MSB is A, LSB is B
*
Hex2Ascii
        PSHA            * store it; work on B first
        ANDA    #$0F    * get LS nybble
        ADDA    #$30    * puts it in ASCII "0" to "?"
        CMPA    #$3A
        BMI     H2A1
        ADDA    #$07    * now it's "A" to "F"
H2A1    TAB             * done with LSB
        PULA
        LSRA            * shift that baby down
        LSRA
        LSRA
        LSRA            * into the lower nybble position
        ADDA    #$30    * puts it in ASCII "0" to "?"
        CMPA    #$3A
        BMI     H2A2
        ADDA    #$07    * now it's "A" to "F"
H2A2    RTS             * done

*******************************************************************************
*
*       SystemInt:       system interrupt routine
*
*           TIMER:      uses TOC4 for control
*
*
*       System interrupt performs the following tasks:
*
*           0.  sets up for next interrupt
*           1.  increment system time
*           2.  decrement "process_ticks".  If zero, pokes
*               BRN (branch never) into pcode_run loop, so that
*               current process exits.
*           3.  deals with LCD print.
*           4.  does PWM stuff.
*           5.  does shaft encoder stuff.
*
SystemInt:
* setup for next interrupt
        LDX     #$1000          * point to register base
        LDD     sysIntCyc       ; default = 500 cycles = .25 millisec = 4 kHz.
        ADDD    TOC4,X          * add TOC4 to D
        STD     TOC4,X          * store back
        BCLR    TFLG1,X %11101111      * clear OC4 for next compare

* increment system time
        LDX     st_lo
        INX
        STX     st_lo
        BNE     si_noinc
        LDX     st_hi
```

```
        INX
        STX     st_hi
si_noinc:

        CLI                             ; enable nested interrupts for IC3 sonar echo
capture

* take analog conversions channels 0 to 2
        LDX     #$1000
        LDAA    #%00010000              ; 4 conversions only
        STAA    ADCTL,X

*wait till done
        BRCLR   ADCTL,X $80 *

        LDAA    ADR1,X                  ; analog 0
        STAA    a0val                   ; store in zero page

        LDAA    ADR3,X                  ; analog 2
        STAA    a2val

        LDAA    ADR2,X                  ; analog 1
        STAA    a1val                   ; store in zero page


        RTI

IC3Int:
        LDX     #BASE
        LDAA    TCTL2,X
        ANDA    #$FC                    ; disable capture (enabled by sonar routine)
        STAA    TCTL2,X
        LDAA    #$00
        STAA    echo_capture            ; tell range routine we've got a ping
        BCLR    TFLG1,X %11111110       * clear IC3 for next capture
        LDD     TIC3,X
        STD     range_time
        RTI



OC5Int:
        LDX     #BASE
        LDAA    #$08            ;clear OC5 flag
        STAA    TFLG1,X
        LDAA    TCTL1,X         ;check polarity of output pwm
        BITA    #$01
        BEQ     zero5
one5    LDD     TI4O5,X         ;it's high
        ADDD    highA3          ;stay high for 'high' cycles
        STD     TI4O5,X
        LDAA    TCTL1,X
        ANDA    #$FE            ; next int change it back to low state
        STAA    TCTL1,X
        RTI
zero5   LDD     TI4O5,X         ;it's low
        ADDD    lowA3           ; stay low for 'low' cycles
        STD     TI4O5,X
        LDAA    TCTL1,X         ;next int change it back to high state
        ORAA    #$01
        STAA    TCTL1,X
        RTI

OC3Int:
        LDX     #BASE
        LDAA    #$20            ;clear OC3 flag
```

```
        STAA    TFLG1,X
        LDAA    portON
        CMPA
        BNE     OC3_pwm
        LDAA    TCTL1,X         ; motor is off
        ANDA    #$EF
        STAA    TCTL1,X         ; clear OC3 output to zero on next compare
        LDD     TOC3,X
        ADDD    #1000
        STD     TOC3,X          ; check every 2msec
        RTI
OC3_pwm
        LDAA    TCTL1,X         ;check polarity of output pwm
        BITA    #$10
        BEQ     zero3
one3    LDD     TOC3,X          ;it's high
        ADDD    highA5          ;stay high on portA pin5 for 'high' cycles
        STD     TOC3,X
        LDAA    TCTL1,X
        ANDA    #$EF            ; next int change it back to low state
        STAA    TCTL1,X
        RTI
zero3   LDD     TOC3,X          ;it's low
        ADDD    lowA5           ; stay low on portA pin5 for 'low' cycles
        STD     TOC3,X
        LDAA    TCTL1,X         ;next int change it back to high state
        ORAA    #$10
        STAA    TCTL1,X
        RTI

OC2Int:
        LDX     #BASE
        LDAA    #$40            ;clear OC2 flag
        STAA    TFLG1,X
        LDAA    stbdON
        CMPA
        BNE     OC2_pwm
        LDAA    TCTL1,X
        ANDA    #$BF
        STAA    TCTL1,X         ; clear OC3 output to zero on next compare
        LDD     TOC2,X
        ADDD    #1000
        STD     TOC2,X          ; check every 2msec
        RTI
OC2_pwm
        LDAA    TCTL1,X         ;check polarity of output pwm
        BITA    #$40
        BEQ     zero2
one2    LDD     TOC2,X          ;it's high
        ADDD    highA6          ;stay high  on portA pin6 for 'high' cycles
        STD     TOC2,X
        LDAA    TCTL1,X
        ANDA    #$BF            ; next int change it back to low state
        STAA    TCTL1,X
        RTI
zero2   LDD     TOC2,X          ;it's low
        ADDD    lowA6           ; stay low  on portA pin6 for 'low' cycles
        STD     TOC2,X
        LDAA    TCTL1,X         ;next int change it back to high state
        ORAA    #$40
        STAA    TCTL1,X
        RTI

* fall through to BadInt

* bad interrupt?  return!
```

```
BadInt: RTI

        Org     $FFC0

        FDB     BadInt  * $FFC0: Reserved
        FDB     BadInt  * $FFC2: Reserved
        FDB     BadInt  * $FFC4: Reserved
        FDB     BadInt  * $FFC6: Reserved

        FDB     BadInt  * $FFC8: Reserved
        FDB     BadInt  * $FFCA: Reserved
        FDB     BadInt  * $FFCC: Reserved
        FDB     BadInt  * $FFCE: Reserved

        FDB     BadInt  * $FFD0: Reserved
        FDB     BadInt  * $FFD2: Reserved
        FDB     BadInt  * $FFD4: Reserved

        FDB     BadInt  * $FFD6: SCI Serial System

        FDB     BadInt  * $FFD8: SPI Serial Transfer Complete
        FDB     BadInt  * $FFDA: Pulse Accumulator Input Edge
        FDB     BadInt  * $FFDC: Pulse Accumulator Overflow
        FDB     BadInt  * $FFDE: Timer Overflow

        FDB     OC5Int  * $FFE0: Timer Input Capture 4/Output Compare 5 (TI4O5)
        FDB     SystemInt * $FFE2: Timer Output Compare 4 (TOC4)
        FDB     OC3Int  * $FFE4: Timer Output Compare 3 (TOC3)
        FDB     OC2Int  * $FFE6: Timer Output Compare 2 (TOC2)

        FDB     BadInt  * $FFE8: Timer Output Compare 1 (TOC1)
        FDB     IC3Int  * $FFEA: Timer Input Capture 3 (TIC3)
        FDB     BadInt  * $FFEC: Timer Input Capture 2 (TIC2)
        FDB     BadInt  * $FFEE: Timer Input Capture 1 (TIC1)

        FDB     BadInt  * $FFF0: Real Time Interrupt (RTI)
        FDB     BadInt  * $FFF2: /IRQ (External Pin or Parallel I/O) (IRQ)
        FDB     BadInt  * $FFF4: /XIRQ (Pseudo Non-Maskable Interrupt) (XIRQ)
        FDB     BadInt  * $FFF6: Software Interrupt (SWI)

        FDB     BadInt  * $FFF8: Illegal Opcode Trap ()
        FDB     BadInt  * $FFFA: COP Failure (Reset) ()
        FDB     BadInt  * $FFFC: COP Clock Monitor Fail (Reset) ()
        FDB     Start   * $FFFE: /RESET
        END
*******************************************************************************
```

Matlab Code for Computing the Experimental RUV Parameters: Calc 5in RUV params.m

This Matlab script calculates the force and moment coefficients for the embedded model used in the experimental RUV controllers. Model A uses a cylinder cross flow drag coefficient of 6 whereas Model B uses a cross flow drag coefficient of 1.

```
%physical constants
rho = 1010;  %fresh water density @ 20C (kg/m^3)
g = 9.81;  %accel grav m/s^2
u0 = .5;
xn = 23.5/39.36;
xt  = -25.5/39.36;
xfin1 = -22/39.36;
xfin2 = -19/39.26;
vmax = .1;  %m/s  max sideslip velocity (for computing RExf)
rmax = 12/57;  %deg/sec
nu = 1e-6;   %m^2/sec  kinematic viscosity

%%some other params
d = 5/39.36;        %max body diam
l = 49/39.36;       %vehicle length
Sw = pi*d*l;    %approx vehicle wetted surface area -cylinder surf area
Ap =l*d;        %hull planform area - rectangle formula
Af = pi*(d/2)^2;
cds = .004;  %see (Paster, 1986)
cdF = .3;    % axial form drag
cda = cds+cdF/(4*(l/d-1));  %total axial drag coeff - skin + form
%
%Model B:
% cylinderCrossflowDragCoeff=1;  %REMUS->1.1 for cylinder crossflow Form drag
% cdc = cds+cylinderCrossflowDragCoeff/(4*(l/d-1));     %total crossflow drag coeff -
skin+form
%ModelA:
cdc = cds+6/(4*(l/d-1));    %total crossflow drag coeff - skin+form  REMUS->1.1 for
cylinder crossflow Form drag
lcp = .65*l-xn;     %center of pressure along hull

rfbar = 4/39.36;    %Distance from vehicle axial center to fin center
Rfin = 5/39.36;     %Distance from vehicle axial center to fin tip
wfin = 3.5/39.36;
tfin = 2.5/39.36;
hfin = 2.5/39.36;
Afin = hfin*(wfin+tfin)/2;
Ar = Afin;      %rudder planform area m^2
As = Ar;        %stern fin planform area
cdf = 1.558;  % fin pressure drag coeff
lfin = 21/39.36;;   %fin moment arm
ARe = (hfin^2/Afin);
cr = 1/(1/2/.9/pi + 1/pi/ARe);      %rudder fin lift coeff
cs = cr;        %stern fin lift coeff

W = 124.7273; %vehicle dry weight N  28lbs
B = 124.73;  %buoyancy force N                    %numerical estimate = 124.66
m = W/g;

xg =    0.00100000;
yg =  -0.0010000;
zg =   0.0508;
xb =   0.00000000;
yb =   0.00000000;
zb =   0.00000000;

Ix =   0.01966000;
Iy =   7.26000000;
Iz =   7.26000000;
Ixy =   0.00000000;
Ixz =  -0.02120000;
Iyz =  -0.02120000;

%added mass parameters: c.f. Blevins formulas and use cylind
%approx for r(x)
```

```
Yvdot = -pi*rho*(d/2)^2*l - pi*rho*(Rfin^2-(d/2)^2+(d/2)^4/Rfin^2)*wfin;  %body+fin
contribution
Zwdot = Yvdot;
Xudot = -.02*4/3*pi*rho*(l/2)*(d/2)^2;
Kpdot = 5*-2/pi*rho*Rfin^4*(xfin2-xfin1);

Mwdot = -(xn^2-xt^2)/2*pi*rho*(d/2)^2 - (xfin2^2-xfin1^2)/2*pi*rho*(Rfin^2-
(d/2)^2+(d/2)^4/Rfin^2);
Nvdot = -Mwdot;
Yrdot = Nvdot;
Zqdot = Mwdot;

Mqdot = -(xn^3-xt^3)/3*pi*rho*(d/2)^2 - (xfin2^3-xfin1^3)/3*pi*rho*(Rfin^2-
(d/2)^2+(d/2)^4/Rfin^2);
Nrdot = Mqdot;

Zqdot = Mwdot; %so that MA is pos definite (see Fossen, p.33)
Yrdot = Nvdot;


%Fin lift terms
Yurf = rho*Ar*cr*lfin;
Yuvf = -rho*Ar*cr;
Zuqf = -rho*As*cs*lfin;
Zuwf = -rho*As*cs;
Muqf = -rho*As*cs*lfin*lfin;
Muwf = -rho*As*cs*lfin;
Nuvf = rho*Ar*cr*lfin;
Nurf = -rho*Ar*cr*lfin*lfin;
%fin forces and moments
Yuudr = rho*Ar*cr;
Zuuds = -rho*As*cs;
Muuds = -rho*As*cs*lfin;
Nuudr = -rho*Ar*cr*lfin;

%Body lift terms
Yuvl = -.5*rho*Ap*.171;  %see Triantafyllou p.140
Zuwl = -.5*rho*Ap*.171;
Muwl = -.5*rho*Ap*.171*lcp;
Nuvl = .5*rho*Ap*.171*lcp;
%combine terms from above
Yur = Xudot+Yurf;
Yuv = Yuvl+Yuvf;
Zuq = -Xudot+Zuqf;
Zuw = Zuwl+Zuwf;
Muq = (Muqf-Zqdot)*75;
Muw = Muwf+Muwl-Zwdot +Xudot;
Nuv = Nuvf + Nuvl + Yvdot -Xudot;
Nur = Nurf + Yrdot;

%Drag terms
I1=pi*d*(xn-xt);
I2=pi*d/3*(xn^3+xt^3);
I3=pi*d*(xn^2-xt^2)/2;
I4=pi*d/4*(xn^4+xt^4);

Xuu = -.5*rho*cda*Sw;
Yvv = 10*-.5*rho*cdc*I1 - rho*cdf*Afin;
Zww = Yvv;
Mww = +.5*rho*cdc*I3 + rho*cdf*lfin*Afin;
Nvv = -Mww;
Kpp = 100*-4*.5*rho*cdf*Afin*rfbar*rfbar*rfbar- 1.0;  %100 is a fudge factor/ additional
1.0 val is finless experiment

Yrr = -.5*rho*cdc*I2 + rho*cdf*lfin*lfin*Afin;
Zqq = -Yrr;
```

```
Mqq = 12.5*-.5*rho*cdc*I4 - rho*cdf*lfin*lfin*lfin*Afin;
Nrr = 10*-.5*rho*cdc*I4 - rho*cdf*lfin*lfin*lfin*Afin;

C_IMPLEMENTATION=0;
%fid = fopen('c:\borlandc\bin\phd\auv\auvREMUSModelCoeff.txt','w');
fid = fopen('C:\Documents and Settings\josseran\Desktop\phd\5in
RUV\auvModelCoeff.txt','w');
if C_IMPLEMENTATION
    fprintf(fid,'const float xg =  %12.8f;\n',xg);
    fprintf(fid,'const float yg = %12.8f;\n',yg);
    fprintf(fid,'const float zg = %12.8f;\n',zg);
    fprintf(fid,'const float xb = %12.8f;\n',xb);
    fprintf(fid,'const float yb = %12.8f;\n',yb);
    fprintf(fid,'const float zb = %12.8f;\n\n',zb);

    fprintf(fid,'const float Ix = %12.8f;\n',Ix);
    fprintf(fid,'const float Iy = %12.8f;\n',Iy);
    fprintf(fid,'const float Iz = %12.8f;\n',Iz);
    fprintf(fid,'const float Ixy = %12.8f;\n',Ixy);
    fprintf(fid,'const float Ixz = %12.8f;\n',Ixz);
    fprintf(fid,'const float Iyz = %12.8f;\n\n',Iyz);


    fprintf(fid,'const float B = %12.8f;\n',B);
    fprintf(fid,'const float W = %12.8f;\n',W);
    fprintf(fid,'const float g = %12.8f;\n',g);
    fprintf(fid,'const float m = %12.8f;\n\n',m );

    fprintf(fid,'const float Nrdot = %12.8f;\n',Nrdot);
    fprintf(fid,'const float Xudot = %12.8f;\n',Xudot);
    fprintf(fid,'const float Yvdot = %12.8f;\n',Yvdot);
    fprintf(fid,'const float Zwdot = %12.8f;\n',Zwdot);
    fprintf(fid,'const float Kpdot = %12.8f;\n',Kpdot);
    fprintf(fid,'const float Mqdot = %12.8f;\n\n',Mqdot);
    fprintf(fid,'const float Mwdot = %12.8f;\n',Mwdot);
    fprintf(fid,'const float Yrdot =  %12.8f;\n',Yrdot);
    fprintf(fid,'const float Nvdot =  %12.8f;\n',Nvdot);
    fprintf(fid,'const float Zqdot = %12.8f;\n\n',Zqdot);

    fprintf(fid,'const float Yur =  %12.8f;\n',Yur);
    fprintf(fid,'const float Yuv = %12.8f;\n',Yuv);
    fprintf(fid,'const float Zuq = %12.8f;\n',Zuq);
    fprintf(fid,'const float Zuw = %12.8f;\n',Zuw);
    fprintf(fid,'const float Muq = %12.8f;\n',Muq);
    fprintf(fid,'const float Muw = %12.8f;\n', Muw);
    fprintf(fid,'const float Nuv = %12.8f;\n\n',Nuv);

    fprintf(fid,'const float Xuu = %12.8f;\n',Xuu);
    fprintf(fid,'const float Yvv = %12.8f;\n',Yvv);
    fprintf(fid,'const float Yrr = %12.8f;\n', Yrr);
    fprintf(fid,'const float Zww = %12.8f;\n',Zww);
    fprintf(fid,'const float Zqq = %12.8f;\n',Zqq);
    fprintf(fid,'const float Kpp = %12.8f;\n',Kpp);
    fprintf(fid,'const float Mww =  %12.8f;\n',Mww);
    fprintf(fid,'const float Mqq = %12.8f;\n',Mqq);
    fprintf(fid,'const float Nvv = %12.8f;\n',Nvv);
    fprintf(fid,'const float Nrr = %12.8f;\n\n',Nrr);

    fprintf(fid,'const float Yuudr = %12.8f;\n',Yuudr);
    fprintf(fid,'const float Zuuds = %12.8f;\n',Zuuds);
    fprintf(fid,'const float Muuds = %12.8f;\n',Muuds );
    fprintf(fid,'const float Nuudr = %12.8f;\n',Nuudr);
else
    fprintf(fid,'xg =  %12.8f;\n',xg);
    fprintf(fid,'yg = %12.8f;\n',yg);
```

```
    fprintf(fid,'zg = %12.8f;\n',zg);
    fprintf(fid,'xb = %12.8f;\n',xb);
    fprintf(fid,'yb = %12.8f;\n',yb);
    fprintf(fid,'zb = %12.8f;\n\n',zb);

    fprintf(fid,'Ix = %12.8f;\n',Ix);
    fprintf(fid,'Iy = %12.8f;\n',Iy);
    fprintf(fid,'Iz = %12.8f;\n',Iz);
    fprintf(fid,'Ixy = %12.8f;\n',Ixy);
    fprintf(fid,'Ixz = %12.8f;\n',Ixz);
    fprintf(fid,'Iyz = %12.8f;\n\n',Iyz);


    fprintf(fid,'B = %12.8f;\n',B);
    fprintf(fid,'W = %12.8f;\n',W);
    fprintf(fid,'g = %12.8f;\n',g);
    fprintf(fid,'m = %12.8f;\n\n',m );

    fprintf(fid,'Nrdot = %12.8f;\n',Nrdot);
    fprintf(fid,'Xudot = %12.8f;\n',Xudot);
    fprintf(fid,'Yvdot = %12.8f;\n',Yvdot);
    fprintf(fid,'Zwdot = %12.8f;\n',Zwdot);
    fprintf(fid,'Kpdot = %12.8f;\n',Kpdot);
    fprintf(fid,'Mqdot = %12.8f;\n\n',Mqdot);
    fprintf(fid,'Mwdot = %12.8f;\n',Mwdot);
    fprintf(fid,'Yrdot =  %12.8f;\n',Yrdot);
    fprintf(fid,'Nvdot =  %12.8f;\n',Nvdot);
    fprintf(fid,'Zqdot = %12.8f;\n\n',Zqdot);

    fprintf(fid,'Yur =  %12.8f;\n',Yur);
    fprintf(fid,'Yuv = %12.8f;\n',Yuv);
    fprintf(fid,'Zuq = %12.8f;\n',Zuq);
    fprintf(fid,'Zuw = %12.8f;\n',Zuw);
    fprintf(fid,'Muq = %12.8f;\n',Muq);
    fprintf(fid,'Muw = %12.8f;\n', Muw);
    fprintf(fid,'Nuv = %12.8f;\n\n',Nuv);

    fprintf(fid,'Xuu = %12.8f;\n',Xuu);
    fprintf(fid,'Yvv = %12.8f;\n',Yvv);
    fprintf(fid,'Yrr = %12.8f;\n', Yrr);
    fprintf(fid,'Zww = %12.8f;\n',Zww);
    fprintf(fid,'Zqq = %12.8f;\n',Zqq);
    fprintf(fid,'Kpp = %12.8f;\n',Kpp);
    fprintf(fid,'Mww =  %12.8f;\n',Mww);
    fprintf(fid,'Mqq = %12.8f;\n',Mqq);
    fprintf(fid,'Nvv = %12.8f;\n',Nvv);
    fprintf(fid,'Nrr = %12.8f;\n\n',Nrr);

    fprintf(fid,'Yuudr = %12.8f;\n',Yuudr);
    fprintf(fid,'Zuuds = %12.8f;\n',Zuuds);
    fprintf(fid,'Muuds = %12.8f;\n',Muuds );
    fprintf(fid,'Nuudr = %12.8f;\n',Nuudr);
end

fclose(fid);
```

References

Allen, B., William, V. S., Prestero, T., 2000, "Propulsion System Performance Enhancements on REMUS AUVs," IEEE *Oceans 2000 Proceedings*, **1**, pp. 1869-1873.

Allen, B., Stokey, R., Austin, T., Forrester, N., Goldsborough, R., Purcell, M., and von Alt, C., 1997, "REMUS: A Small, Low Cost AUV; System Description, Field Trials and Performance Results," IEEE *Oceans 1997 Proceedings*, **2**, pp. 994-1000.

Arulampalam, S., Maskell, S.,Gordon, N., and Clapp, T., 2002, "A Tutorial on Particle Filters for On-line Non-linear/Non-Gaussian Bayesian Tracking," IEEE *Transactions on Signal Processing*, **50**, pp. 174-188.

Autosub, http://www.soc.soton.ac.uk/PR/Autosub.html

Bagotronix, http://www.bagotronix.com/dosstamp.html

Bishop, R. H., 2002, *Stochastic Estimation and Control*, The University of Texas at Austin, Department of Aerospace Engineering, Lecture Notes for ASE 381P.8.

Buckner, G. D., Fernandez, B., and Masada, G., 2002, "Intelligent Bounds on Modeling Uncertainty: Applications to Sliding Mode Control," IEEE *Transactions on Systems, Man, and Cybernetics-Part C: Applications and Reviews*, **32**, pp. 113-124.

Cristi, R., Papoulias, F., and Healey, A. J., 1990, "Adaptive Sliding Mode Control of Autonomous Underwater Vehicles in the Dive Plane," IEEE *Journal of Oceanic Engineering*, **15**, pp152-160.

Daum, F. E. and Huang, J., 2002, "Mysterious Computational Complexity of Particle Filters," *Proceeding of SPIE*, Signal and Data Processing of Small Targets 2002, Oliver E. Drummond; Ed., **4728**, pp. 418-426

DeCarlo, R. A., Zak, S. H., and Matthews, G. P., 1988, "Variable Structure Control of Nonlinear Multivariable Systems: A Tutorial," *Proceedings of the* IEEE, **76**, pp. 212-232.

Doucet, A., de Freitas, N., and Gordon, N., Eds., 2001, *Sequential Monte Carlo Methods in Practice*, Springer-Verlag, 2001.

Dorf and Bishop, Control

Feldman, J., 1979, "Revised Standard Submarine Equations of Motion," Report No. DTNSRDC/SPD-0392-09, David W. Taylor Naval Ship Research and Development Center.

Fernandez, B., and Buckner, G.D., 1995, "Statistica: Artificial Neural Networks for Statistical Bounding of Data," *Proceedings of the 4th IEEE Conference on Control Applications*, Albany, New York, pp. 554-559.

Fernandez, J. E., Christoff, J. T., and Cook, D. A., 2003, "Synthetic Aperture Sonar on AUV," IEEE *Oceans 2003 Proceedings*, **3**, pp. 1718-1722.

Fong, D. A., and Jones, N. L., 2006, "Evaluation of AUV-based ADCP Measurements," *Limnology and Oceanography: Methods*, **4**, pp. 58-67.

Fossen, T., 1994, *Guidance and Control of Ocean Vehicles*, John Wiley and Sons, Inc., New York.

Fox, R.W. and McDonald, A. T.,1992, *Introduction to Fluid Mechanics*, John Wiley and Sons, Inc., New York.

Grenon, G., An, P. E., Smith, S. M., and Healey, A. J., 2001, "Enhancement of the Inertial Navigation System for the Morpheus Autonomous Underwater Vehicles," IEEE *Journal of Oceanic Engineering,***26**, pp. 548-560.

Gordon, N., Salmond, D., and Smith, A. F. M., 1993, "Novel Approach to Non-linear and Non-Gaussian Bayesian State Estimation," IEE *Proceedings F*, **140**, pp. 107-113.

Hoerner, S. F., 1965, *Fluid Dynamic Drag*, Hoerner Fluid Dynamics, Midland Park, New Jersey.

Hydroid, http://www.hydroidinc.com/

Isidori, A. 1989, *Nonlinear Control Systems: An Introduction, 2nd Edition*, Springer Verlag, New York.

Jones, D.A, Brayshaw, I.B., Barillon, J.L., and Anderson, B., 2002, "The Calculation of Hydrodynamic Coefficients for Underwater Vehicles," Defense Science and Technology Organization, Australia, www.dsto.defence.gov.au/ publications/2519/DSTO-TR-1329.pdf

Lewis, F., 1986, *Optimal Estimation with an Introduction to Stochastic Control Theory*, John Wiley and Sons, Inc., New York.

Lewis, F., 1992, *Applied Optimal Control and Estimation*, Prentice Hall, Englewood, New Jersey.

Loria, A., Fossen, T. I., and Panteley, E., 2000, "A Separation Principle for Dynamic Positioning of Ships: Theoretical and Experimental Results," IEEE *Transactions on Control Systems Technology*, **2**, pp. 332-343.

Lu, Xiao-Yun, and Spurgeon, S. K., 1999, "Robustness of Static Sliding Mode Control for Non-linear Systems," *International Journal of Control*, **72**, pp. 1343-1353.

Martin, F., 1998, "The Mini Board Technical Reference," The Media Laboratory, MIT, http://www.cs.uml.edu/~fredm//cher/projects/miniboard/docs/

Marion, J. B. and Thornton, S. T., 1988, *Classical Dynamics of Particles and Systems*, 3rd Edition, HBJ Publishers, Inc., San Diego, CA.

Mindell, D. and Bingham, B., 2001, "New Archeological Uses of Autonomous Underwater Vehicles," IEEE *Oceans 2001 Proceedings*, **1**, pp. 555-558.

Ogata, K., 1990, *Modern Control Engineering, 2nd Ed.,* Prentice-Hall, Inc., New Jersey.

Paster, D. L., 1986, "Importance of Hydrodynamic Considerations for Underwater Vehicle Design," ," IEEE *Oceans 1997 Proceedings*, **18**, pp. 1413-1422.

Perruquetti, P., Borne, P., and Richard, J. P., 1997, "A Generalized Regular Form for Sliding Mode Stabilization of MIMO Systems," IEEE *Proceedings of the 36th Conference on Decision and Control*, San Diego, CA, pp. 957-961.

Prestero, Timothy, 2001, "Verification of a Six-Degree of Freedom Simulation Model for the REMUS Autonomous Underwater Vehicle," MS Thesis, Massachusetts Institute of Technology, Joint Program in Applied Ocean Science and Engineering.

Peterson, R. S., Nguyen, T. C., and Rodriquez, R. R., 1994, "Motion Minimization of AUVs For Improved Imaging Sensor Performance Beneath a Seaway," IEEE

Proceedings of the 1994 Symposium on Autonomous Underwater Vehicle Technology, 1994, AUV '94, Cambridge, MA, pp. 247-254.

Roup, A. V. and Humphreys, D. E., 2005, "Bluefin Robotics AO SAS 12.75" UUV: Hydrodynamic and Autopilot Design and In Water Validation," VCT Tech Memo 05-05, Vehicle Control Technologies, Inc., Reston, VA.

Schilling, R.J. and Harris, S.L., 2000, *Applied Numerical Methods for Engineers,* Brooks/Cole, California.

Silpa-Anan, C. and Zelinsky, A., 2001, "Kambara: Past, Present and Future," *Proc. 2001 Australian Conference on Robotics and Automation*, Sydney, November 14-15, pp. 61-66.

Slotine, J. J., and Li, 1991, *Applied Nonlinear Control*, Prentice Hall, Englewood, New Jersey.

Strang, G., 1988, *Linear Algebra and Its Applications*, 3rd Ed., Academic Press, New York.

SNAME - Society for Naval Architects and Marine Engineers, 1964, "Nomenclature For Treating the Motion of a Submerged Body Through a Fluid," *Society for Naval Architects and Marine Engineers Technical and Research Bulletin*, No. 1-5.

Triantafyllou, M. S. and Hover, F. S., 2003, *Maneuvering and Control of Marine Vehicles*. Available online: http://ocw.mit.edu/OcwWeb/Ocean-Engineering/13-49Fall-2004/LectureNotes/index.htm

Utkin, V. I., 1978, *Sliding Modes and Their Application in Variable Structure Systems*, MIR, Moscow.

Yoerger, D. R. and Slotine, J-J. E., 1985, "Robust Trajectory Control of Underwater Vehicles," IEEE *Journal of Oceanic Engineering*, **10**, pp. 462-470.

Yoerger, D. R., Newman, J. B., and Slotine, J-J. E., 1986, "Supervisory Control System for the Jason ROV," IEEE *Journal of Oceanic Engineering*, **3**, pp. 392-400.

Yue, D. K. P., 2004, *Marine Hydrodynamics*, MIT, Department of Ocean Engineering, Lecture Notes for Ocean Engineering, 13.021. Available online: http://ocw.mit.edu/OcwWeb/Ocean-Engineering/13-021Marine-HydrodynamicsFall2001/LectureNotes/index.htm

Vita

Tim Josserand was born in Austin, TX on March 11th, 1970. His parents are Mike Josserand and Hope Zelasko. He graduated Summa Cum Laude from Metropolitan State College of Denver with a B.S. in Applied Mathematics and a minor in Physics. At the University of Texas at Austin he graduated in May 2000 with a M.S. in Mechanical Engineering. He subsequently acquired full-time employment with the University at the Applied Research Laboratories in the Advanced Technology Lab. While working as an Engineering Scientist on problems in applied underwater acoustics and robotics, he pursued his Ph.D. part-time at UT until fulfillment.

Permanent address: 901 Edgecliff Terrace, Austin, TX 78704

This dissertation was typed by Tim Josserand.